7-24-1986

Linkage kinematics sketchpad

Michael St. Jacques

Rochester Institute of Technology
School of Computer Science & Technology

Linkage Kinematics Sketchpad

by

Michael St.Jacques

A thesis, submitted to
the Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

Approved by:

Guy Johnson 7/24/86
Professor Guy Johnson, Chairman Date

Jack Hollingsworth 7/31/86
Professor Jack Hollingsworth Date

John L. Ellis
Professor John L. Ellis Date
Graduate Studies Committee

James Carbin 7/24/86
Professor James Carbin Date

Abstract

During the design and creation of linkage-type mechanisms, visualization of linkage motion is extremely important. However, there does not appear to be a commercially available computer package for accomplishing visualization interactively. Most linkage design packages allow animation of linkage motion only after tedious part description (and debug) using cryptic input codes.

The main thrust of this work has been the development of a prototype interactive graphics (CAD) system aimed at visualizing the motion and mobility of linkage-type mechanisms. The program is called the Linkage Kinematics Sketchpad (LKSP). It is a 2-D color graphics program which allows the user to describe a limited set of linkages (limited by a simplified kinematics analysis procedure) and interactively drive the linkage through its inherent motion cycle (or parts thereof) to visualize mobility.

First, a theoretical investigation of previous work in motion analysis and display of animation is presented. This is followed by a description of the LKSP program and an evaluation of the software by this author and others more familiar with linkage design. The system design appears to be adequate, and the software is correct with linkage motions as required. As a result of this work the usefulness of this approach has been determined, and a reasonable methodology has been established. Also, problem areas have been defined, and potentially fruitful areas for future work have been identified. LKSP offers a unique approach to planar linkage design with the most desirable features being the interactive user-computer interface, the ability to create linkages with ease, and the ability to observe linkage motion and potential interference.

The most commonly cited shortcoming was the limited set of linkage components which LKSP can handle. Also, there were some aspects of the motion animation which were improved as a result of the user evaluations. Suggestions for future extensions include more user control over the motion animation and more precise input of linkage dimensions.

Key Words and Phrases

Linkage Kinematics
Computer Graphics
Linkage Analysis
Linkage Design
Linkage Synthesis
Mechanisms
Planar Linkages
Computer-aided-design, CAD

Acknowledgements.

There are many who helped make this work possible. First, many thanks to Bernie Mc Cabe for valuable assistance in preparation of the video tape illustrating linkage creation and animation which I used during my defense. Also, many thanks to Ray Curtin for conducting the extensive on-line literature search. In addition, grateful thanks to Mark Erdrich and Susan Linder for much help with linkage kinematics and the loan of a number of valuable references. Also, thanks to Bruce Schwager and Ray Dennis for help with VMS and again to Mark Erdrich, Susan Linder and Bruce Schwager for evaluating the LKSP software. Special thanks to JoAnne Weber for proof reading and careful review. Also, Many thanks to my graduate review committee at R.I.T., Professor Guy Johnson, Chairman, Professor Jack Hollingsworth, Professor John L. Ellis, Graduate Studies Committee and Professor James Carbin for reviewing this work, and guiding it to a fruitful conclusion. In addition, many thanks to my friend Frank Pachla for always pointing out "the beauty part". Finally, grateful thanks to my loving wife Mary Lynne for her continual encouragement, proof reading, and many late suppers.

Table of Contents.

	page
1. Introduction and Background.	1
1.1 Problem Statement.	1
1.2 Previous Work.	2
1.2.1 Mechanism Engineering, Background.	2
1.2.2 Linkage Design Process.	3
1.2.3 Literature Search, Strategy.	7
1.2.4 Summary of Previous Work.	8
1.3 Theoretical and Conceptual Development.	14
2. Project Description.	33
2.1 Functional Description.	33
2.1.1 Functions Performed.	34
2.1.2 Limitations and Restrictions.	35
2.1.3 User Inputs and Outputs.	37
2.1.4 System Files.	50
2.2 System Description.	55
2.2.1 System Organization.	55
2.2.2 System Data Flow.	61
2.2.3 Equipment Configuration.	61
2.2.4 Implementation Tools.	74
3. Evaluation of LKSP, Conclusions and Recommendations.	75
3.1 User/computer Interface.	76
3.2 Usefulness of Unique Features.	76
3.3 Operating Efficiency.	87
3.4 The Hardware Used.	87
3.5 Suggestions for Future Extensions.	88
4. Bibliography.	89

Appendices.

Appendix I Glossary.
Appendix II Listing, Main Program.
Appendix III Listing, Prep Module.
Appendix IV Device Driver and
Core Graphics Procedures.

Figures. page

Figure 1 1	Examples of Looped and Non-looped Linkages.	4
Figure 1-2	Proposed Linkage Components, Drivers and Non-drivers.	16
Figure 1 3	Example Linkages.	17
Figure 1-4	Rotating Driver.	19
Figure 1 5	Translating Driver.	21
Figure 1-6	Dyad-1.	23
Figure 1 7	Dyad-2.	25
Figure 1-8	Rotating Guide.	27
Figure 1-9	Oscillating Slider.	29
Figure 1 10	Slider.	31
Figure 2-1	Linkages Which Cannot Be Described Within LKSP.	36
Figure 2-2	Screen Layout.	38
Figure 2-3	Menu Layout.	39
Figure 2-4	LKSP Components, Rotating Driver, and Two Simple Dyads.	41
Figure 2-5	LKSP Components, Rotating Driver, Oscillating-Slider, Dyad-2 and Dyad-1.	42
Figure 2-6	LKSP Components, Rotating Driver, Rotating Guide, Slider and Dyad-1.	43
Figure 2 7	Rotating Driver with All LKSP Components.	44
Figure 2-8	Rotating Driver with Dyad-1, in Motion, No Trace.	46

Figure 2-9	Rotating Driver with Rotating Guide, in Motion, Trace "on", 2 I-panels.	47
Figure 2-10	Rotating Driver with Dyad-2, in Motion, Trace "on".	48
Figure 2-11	Rotating Driver with Dyad-2 and Oscillating Slider, in Motion, Trace "on", Linkage Binding.	49
Figure 2-12	Translating Driver with Dyad-1, in Motion, Trace "on", 2 I-panels.	51
Figure 2-13	Translating Driver with Rotating Guide in Motion, Trace "on", 2 I-panels.	52
Figure 2-14	System Files Hierarchy.	53
Figure 2-15	System Files Structure.	54
Figure 2-16	System Module Hierarchy.	71
Figure 2-17	System Data Flow, Top Level.	72
Figure 2-18	Equipment Configuration.	73
Figure 3-1	Rotating Driver with Rotating Guide, in Motion, 1 Degree Rotation Increment, Trace "on".	78
Figure 3-2	Translating Driver with Dyad-2, in Motion, Trace "on".	79
Figure 3-3	Rotating Driver with Slider (Crank-Slider), in Motion, Trace "on", Only Tracer Path Displayed.	80
Figure 3-4	Rotating Driver with Dyad-2, in Motion, Trace "on", Only Tracer Path Displayed.	81
Figure 3-5	Rotating Driver with Rotating Guide, in Motion, Trace "on", Only Tracer Path Displayed.	83

Figure 3-6	Rotating Driver with Oscillating Slider and Rotating Guide, in Motion, Trace "on", Only Tracer Path Displayed.	84
Figure 3-7	Rotating Driver with Dyad-1, in Motion, Trace "on", Mode of Assembly Reverses.	85
Figure 3-8	Rotating Driver with Dyad-2, in Motion, Trace "on", Only Tracer Path Displayed, Linkage Binding.	86

Tables. page

Table 1 1	Trigonometric Functions and Coordinate Transforms Used in Kinematics Algorithms.	18
Table 1-2	Rotating Driver Algorithms.	20
Table 1-3	Translating Driver Algorithms.	22
Table 1-4	Dyad-1 Algorithms.	24
Table 1-5	Dyad-2 Algorithms.	26
Table 1-6	Rotating Guide Algorithms.	28
Table 1 7	Oscillating Slider Algorithms.	30
Table 1-8	Slider Algorithms.	32
Table 2-1	Procedures Accessed by Main Software Module in LKSP.	56
Table 2-2	Procedures Accessed by Prep.	57

Table 2-3	Procedures Accessed by makeMenu Software Module.	58
Table 2-4	Procedures Accessed by DisplayMenu Software Module.	59
Table 2 5	Procedures Accessed by makeAccMenu Software Module.	60
Table 2-6	Miscellaneous Software Modules.	62
Table 2-7	Miscellaneous Software Modules, Continued.	63
Table 2-8	Function of the "do" Software Modules.	64
Table 2-9	Procedures Accessed by "do" Routines, DisplayAccMenu Modules.	65
Table 2-10	Procedures Accessed by "do" Routines, "get" Modules.	66
Table 2-11	Procedures Accessed by "do" Routines, Support Procedures, Part 1.	67
Table 2-12	Procedures Accessed by "do" Routines, Support Procedures, Part 2.	68
Table 2 13	Procedures Accessed by "do" Routines, Support Procedures, Part 3.	69
Table 2-14	Code Summary.	70

1. Introduction and Background.

1.2 Previous Work

In this section, a summary of relevant background on mechanisms engineering (1.2.1) and the linkage design process (1.2.2) is offered. A view of the complete linkage design process is presented in order to create an appreciation of how and where motion visualization is useful to the linkage designer.

Next, the strategy used for a comprehensive library search (1.2.3) and the results of the search relative to analysis as well as graphics and animation (1.2.4) is presented. Kinematic analysis is discussed in detail because it forms the topological basis for the final selection of mechanism types used in the project.

1.2.1 Mechanism Engineering, Background

The following commonly accepted definitions are offered to orient readers unfamiliar with pertinent engineering jargon. These definitions are also summarized in the glossary of Section 1.4.

The term "mechanism" was defined in 1875 by Reuleaux (1)* as "a combination of rigid or resistant bodies so formed and connected that they move upon each other with definite relative motion." Since the publication of Reuleaux's classic theoretical treatise on kinematics of machinery, this definition has remained essentially unchanged.

The purpose of a mechanism is to transmit motion from one body to another. A "machine" is comprised of one or more mechanisms typically configured to work together to perform some overall machine function(s).

Mechanisms are frequently categorized into three basic types (2)

1. gear systems, or gear trains in which meshed gears are the primary motion transmitters and the motion is rotary.

2. cam systems, consisting of cams and followers, in which the rotational motion of the cam is transformed into translation of the follower.

* Numbers in parentheses are literature references in the Bibliography.

3. linkages, consisting of "links" or bodies (usually, but not always assumed to be rigid) connected together, in which the motion of one body (or more than one) influences the motion of the others in some desired fashion (e.g. see Figure 1 1).

Each of the above mechanism types represents a fairly extensive subset with highly specialized design and analysis techniques. Only linkages are considered further in this work. In fact only a small (but very useful) subset of linkages is included, namely "planar" linkages in which all motion is planar or in parallel planes (i.e. 2-D) as opposed to spatial (i.e. 3-D) linkages.

Another linkage categorization consists of whether or not the "chain" of connected rigid bodies forms a loop (see Figure 1 1). Looped mechanisms were once the most common type (e.g. the slider-crank or "crank shaft" in an auto engine). The advent of nonlooped mechanisms is certainly apparent in the form of robotic manipulators and servo-driven disk heads.

1.2.2 Linkage Design Process

The linkage design process generally consists of the following steps

1. Identification of design objectives.
2. Preliminary ideas.
3. Refinement.
4. Analysis.
5. Decision.
6. Implementation.

Identification of design objectives (step 1) is related to linkage motion and to the forces producing (or produced by) motion. Often, these can be handled analytically. Design objectives of this type are generally categorized (3) into

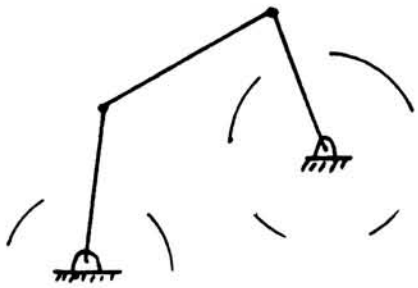
a. Kinematics (i.e. related to linkage motion),

   1) Guidance of a rigid body (e.g. a bucket loader),

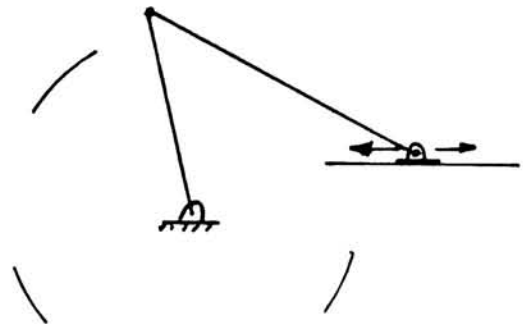
   2) Path generation (e.g. a sewing machine needle),

   3) Function generation (e.g. throttle linkage),

Looped.

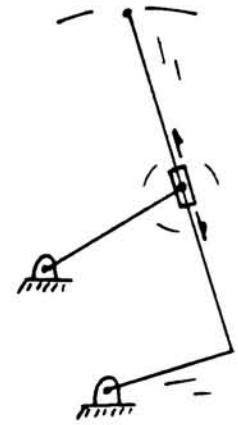


Basic 4-Bar Linkage.



Offset Slider Crank.

Crank, Rotating Guide.



Crank, Oscillating Slider

Nonlooped.

Figure 1 1 Examples of Looped and Non-looped Linkages.

b. Kinetics (related to forces producing motion)

1) Vibrations (e.g. reciprocating engines)

2) Dynamic Loads (e.g. disk head on arm)

3) Motion-time response (e.g. control valve)

4) Static forces (e.g. a scissors jack)

5) Impact forces (e.g. a punch press)

6) Elasticity (e.g. flexible linkages)

Another category includes an array of practical constraints normally included in design objectives (4). These include

c. Practical Constraints

1) Accessibility for Maintenance

2) Overall Weight

3) Fabrication Cost

4) Design Completion Deadline

5) Existing Space Restrictions

A complete design objective is usually some trade-off between kinematic and kinetic goals and practical constraints.

Motion visualization plays a important role in all designs, regardless of goals, but especially in understanding kinematics and the performance of the linkage with regard to practical constraints.

Preliminary ideas (step 2) include idea generation (brainstorming) based on the design objectives. Few (if any) ideas are eliminated at this point. Motion visualization plays an important role here in the study of basic feasibility.

Ideas generated in step 2 are then refined (step 3) by limited analysis, past experience, and other criteria. If possible, ideas are eliminated at this point. Depending on goals, motion visualization may play a vital role here to further enhance feasibility studies.

Analysis (step 4) of remaining design concepts takes place, again with the idea of concept elimination through evaluation of trade-offs. Depending on goals, visualization of linkage motion may be important here to enhance the designer's conceptual understanding of how well the linkage meets the kinematics, kinetics, and other design goals.

Generally, kinematic analysis is done before other types of analysis (e.g. kinetics) because a complete understanding of linkage motion is usually required first. Also, physical design space may be limited and interference checking via kinematic analysis is often needed.

The reverse of analysis is synthesis wherein a design objective is specified, and the geometry of the linkage is determined. There are normally two synthesis steps, "type synthesis" wherein the category of the proposed linkage is determined, and "dimensional synthesis" in which actual dimensions are specified. A linkage is "synthesized" or created through a set of rules to meet the specification within some predetermined limit.

For example, a design objective such as path generation is specified as a set of "precision points" or locations on the desired path where any linkage attributes (e.g. position, velocity or acceleration) are to be met. Synthesis has also received its share of attention in the engineering literature.

After analysis (at least to some degree of completeness), a decision (step 5) is made regarding the final design. In reality, the decision is probably made after looping through the above four steps several times. The time required to get decisions can be shortened with good visualization tools.

Finally, during implementation (step 6), the selected design idea becomes reality. If all works as planned, the process is complete. More likely, some unforeseen event such as a last minute change in another part of the machine or a missed obstruction to motion will require retrofit. Visualization of linkage motion may be extremely important at this stage to compare the original design to possible retrofit candidates. Interference checking can also be vital here.

In summary, visualization of linkage motion can play an important and useful role at any stage of the design process.

1.2.3 Literature Search, Strategy

Two on-line literature data base systems were accessed from information services organizations to perform the computerized library search of this topic.

These are Compendex (formerly the Engineering Index) offered by Engineering Information of New York City, and ISMEC (Information Service in Mechanical Engineering and Consulting) offered by Cambridge Scientific Abstracts of Bethesda, Maryland.

Because the topic of linkage design using CAD is so specialized, it was felt that no other data base systems would yield further, useful information (6).

The keywords structure used for the search was formed as a logical (and/or) combination of user-defined words. The data base is searched for the required combination of words (in any order, but logically combined as specified).

The information services organizations accessed to perform this search "read" the entire abstract of each article (not just the title) resulting in a very complete list of citations. Data bases were searched back in time to their creation.

The following keyword structure was used

kinematics.
AND

analysis,
design,
synthesis.
AND

mechanisms,
linkages,
manipulators,
machines,
machinery.
AND

computer.
computer-aided,
computer-aided-design,
CAD,
C.A.D,
C. A. D.

where columns are logically "ANDed" with other columns. Within a column the words are "ORed". So an article that mentions, in its abstract, at least one word from all four columns is accepted. All others are rejected. This search technique yielded a total of 52 articles. A review of the abstracts eliminated about twenty of these because their thrust was primarily synthesis. Copies of the remaining articles were obtained and reviewed. These appear in the bibliography along with literature obtained through other sources.

It appears, as expected, that analysis and synthesis have received the most attention in the literature. This is usual for engineering applications, with interactive graphics lagging behind.

About 70% of the articles reviewed (both from the search and elsewhere) deal with analysis in some form or another. Another 16% are survey articles offering either a general review of the reported work in the field or specifics about a current software offering.

Some of the articles were unique in some way, and did not fit into any other category. Only a few articles deal with specifics of motion visualization or interactive graphics. The primary thrust of most of the literature is analysis.

1.2.4 Summary of Previous Work

In the bibliography (Chapter 4) articles are arranged by topic in the order discussed in section 1.2.4. Within a topic, references are arranged chronologically. As mentioned, analysis and synthesis have received the most attention in the engineering literature. If the linkage is nonlooped, the analysis method is often based on matrix coordinate transform techniques (6 to 10) wherein the relative linkage motions are transformed into absolute motion with respect to a base coordinate system.

The algorithms are basically repetitive matrix multiplications, with at least one multiplication for each degree of freedom for each link. Often, a sequential procedure is possible from the base or driver component through each link, one by one, to the last component (e.g. the end effector). For every new position of the driver (known) new positions of every link are computed.

Recent work on analysis techniques for nonlooped linkages has centered on modification for microcomputers (11) and dynamic, real time computation and display with sophisticated solids-modeling graphics (12).

The earliest analysis methods for looped planar linkages were based on geometric construction and repetitive reconstruction of the linkage on a drafting table (13, 14). Oftentimes, cardboard or wood models of the proposed linkage were constructed and used to check kinematics (3).

Standard vector analytical procedures were used in conjunction with geometric construction and eventually became widely used. Many tedious hours were spent hand calculating kinematic data on complex linkages (38). These techniques are a far cry from the computer based techniques available today.

Despite the availability of computer based techniques (and computers), Prentis (37) has recently presented an article on a method of kinematic and dynamic analysis using a programmable calculator.

Computer analysis methods for looped linkages (15 to 32) are based on matrix coordinate transform techniques similar to the nonlooped case with the following modification.

The transformation matrices are no longer simply multiplied together for each link, but the product of transformation matrices for each complete loop is set equal to the identity matrix.

A solution to the resulting set of simultaneous, non-linear equations is obtained (e.g. by an iterative procedure similar to that described in Reference 15) and the next loop is treated a similar fashion.

Some of the newly computed positions for one loop are input movements to connected loops. All loops in the linkage are solved either simultaneously or sequentially.

Obviously, solution algorithms must understand which links belong to which loops and how many loops there are. Therefore one of three approaches is possible

1. The analysis package must be restricted to a prescribed topology.

2. The user must identify the loops.

3. An algorithm must be incorporated to identify kinematic loops (topology) from other input information.

One recent design package for the IBM PC (44) is restricted to a subset of 4-bar linkages in which at least one bar can make a full 360 degree rotation. This fits

well into category 1 above. A technique similar to category 2 above was described by Shu and Radcliffe (2) in 1978.

The automated topology referred to in category 3 above has received some attention in the literature (33 to 35). The DRAM program (Dynamic Response of Articulated Machinery), completed in 1969, appears to be the first comprehensive attempt at both topology and analysis of arbitrary 2-D linkages (32).

IMP (Integrated Mechanisms Program), first publicized in 1971 (33), was the first package to handle analysis of (topologically) arbitrary spatial linkages. Following in 1973 was the ADAMS program (Automated Dynamic Analysis of Mechanical Systems), also capable of handling spatial systems with arbitrary topology (32, 46, 48).

Detailed presentations of matrix methods for topological and kinematic analysis of spatial linkages were presented by Uicker et. al. (15 to 19). A decent tutorial on synthesis techniques has been offered by Chase (36).

The main differences among these computer packages are connected to the choice of coordinate system and the method of solving the resulting equations (32). Gaussian elimination is used for coordinate systems which yield small, dense sets of equations (DRAM).

Sparse matrix techniques are used for cases in which a large, sparse set of equations results (ADAMS). Speed of execution is strongly dependent on how well the coordinate system and solution method match the problem.

Because of the specifics of the solution algorithm, some packages perform better than others for certain linkage types. For example, an algorithm that carries inertial and force data through the computational steps is likely to run slower than one that handles only kinematics.

Up until the mid-1970s, the main thrust of attempts at computer application to linkage design was analysis (or synthesis) as opposed to adequate analysis with user-friendly software.

The earliest surveys of package capability (38, 39), published in 1974 and 1978 respectively, showed that most of the analysis programs evolved from university research. Animation of kinematics was sometimes provided as graphical output if the user was fortunate enough to have the right graphics display terminal.

One of the earliest efforts to simplify the whole design

cycle through the use of graphics came out of DuPont's Engineering Development Laboratory. This work was reported by Ladd (4) in late 1976.

Input of linkage data with DuPont's program was still accomplished with a "proprietary graphics input language." However, the user was allowed to input geometric constraints to movement (boundaries) along with the linkage. Animation of the linkage with resulting path curves was also possible.

The earliest attempt to use interactive graphics for linkage design was reported by Rubel and Kaufman (40, 41) of M.I.T. in 1977. They described the KINSYN program (KIN-ematic SYN-thesis) which they called a "Human-Engineered System for Interactive Computer Aided Design of Planar Linkages".

In KINSYN, the user's interaction with the computer was completely graphical rather than through problem-oriented language statements typed on a keyboard. A path of required motion (up to 5 precision points) was drawn by the user on a tablet. The synthesis capability then took over and produced a linkage.

KINSYN was programmed on a "customized in-house mini computer system" (42) at M.I.T. and was therefore considered to be primarily a research toy. The LINCAGES program (L-inkage IN-teractive C-omputer A-nalysis and G-raphically E-nhanced S-ynthesis) was described by Erdman et al (42, 43) in 1977 as a time-sharing package with the same capabilities as KINSYN, and accessible to anyone with a Tektronix 4010 storage-tube and a telephone modem. Versions of LINCAGES are available today with interactive graphics input.

Today there are many more linkage design and synthesis programs available. The most recent surveys (46, 49) report at least 18 programs known to be in use for various types of analysis and synthesis. These programs can be run on various systems including everything from personal computers to mainframes.

One recent package, DADS (D-ynamic A-nalysis and D-esign S-oftware) offered by Computer Aided Design Software, Inc. (47) in 1985 includes a "smart" pre-processor which automatically deletes obvious typos and syntax errors in the input code where intent can be uniquely resolved.

DADS also includes a language-specific editor to ease the task of input code preparation. Post-processors are also available to intelligently manipulate and handle text and

graphics output.

Like many recent packages and updates of earlier software, "animation capability" is routinely available. Usually, animation capability means the entire linkage is moved through its motion range with tracing of selected points to show paths of interest. Also, many packages use graphics to display pertinent engineering data, usually with time (e.g. velocity, acceleration or constant angular velocity driving force).

Graphics is also used for input checking. The program's understanding of the linkage, which the user has described with a problem-oriented input language, is drawn on a CRT. Discrepancies are rectified by resubmitting a revised input data file.

Strangely enough, there does not appear to be evidence that any of the analysis packages use interactive graphics for input. Another recent survey by Dawson, appearing in early 1985 (48) states that turnkey CAD suppliers are currently writing software to interface their graphics systems to DRAM and ADAMS. Interfaces would automatically create the geometric input data, required by DRAM or ADAMS, from the CAD system's existing geometric data base. Additional data for analysis such as loads and constraints must still be input separately.

Interfacing allows a part, previously described through user-friendly interactive graphics to a CAD system, to be analyzed, thus avoiding the tedium of problem oriented input coding. In today's fast moving design world, engineers who are called-upon to use many different computer packages certainly need integrated software.

Dawson further states that six CAD houses have announced interfaces. These include Applicon, Computer-Vision, Control Data, Intergraph, McAuto and Sperry. It is probable that interfacing is feasible today.

Other recent surveys (49, 52) from 1985 indicate that

1. Several groups have begun to market micro-computer based analysis software.

2. Robotics animation, work space prediction, dynamic response and interference checking capabilities are becoming more sophisticated.

3. Integration of mechanism analysis and synthesis software with CAD drafting, finite-element programs and simulation software has begun.

One specific effort aimed at integration of software was reported by Mittelstadt et al (52) in 1985. This effort is being carried out at the University of Minnesota, and is sponsored by Control Data Corporation. The aim was to tie three major components together into one integrated tool. Components of the prototype system will include LINCAGES for synthesis, DRAM for analysis and ICEM DESIGN/DRAFTING from Control Data Corporation for design and drafting.

Erdman (52) reports that the future promises more and improved capabilities as further software integration takes place. More sophisticated graphics capability and faster hardware with solids modeling capability are also expected. Artificial intelligence techniques, in particular expert systems for synthesis, are likely to be developed and incorporated with linkage design software. As the expert system heuristics are packaged together, it is probable that earlier work will be incorporated.

One example of an earlier development that would be useful for a future expert system is Tuttle's 1967 effort (50). Tuttle compiled a data base of successfully used mechanisms categorized by type of output motion (heuristics).

Also, expert systems might require algorithms for translating mechanisms being considered during a synthesis session from internal form (graph representation) to a line sketch. This type of program would be handy for graphically updating an expert system user on the system's current "thinking." This capability was reported by Olson et al (51) in 1985 in their paper "Automatic Sketching of Planar Kinematic Chains."

The capability of interactively "sketching" a linkage has received little publicity (and perhaps little attention). The only direct reference to sketching in this literature search (aside from the earlier mentioned KINSYN synthesis program) came from Erdman and Riley (46).

In mid-1985, these researchers reported results of their experiences at the University of Minnesota in teaching machine design courses with commercially offered courseware (i.e. software designed specifically to teach something). The courseware system which they used is called Plato, and is offered by Control Data Corporation on a timesharing basis via telephone connection. Plato offers hundreds of tutorial sessions on various topics.

One portion of the Plato courseware is called the "4-bar sketchpad." This lesson allows the student to define an arbitrary four-bar linkage with either a touch screen or arrow keys on the keyboard. IBM or Zenith personal computers in terminal emulation mode are supported.

After the linkage is described by the student, the Plato system animates the linkage by tracing the paths of moving points. Erdman and Riley indicated excellent results with the Plato system.

This concludes the results of the literature search. It seems apparent that the concept of an interactive sketchpad coupled with the more common motion visualization capability via animation would be useful even to experienced designers and even more useful to inexperienced engineers who are trying to use many different computer packages.

1.3 Theoretical and Conceptual Development

In essence, the Linkage Kinematics Sketchpad (LKSP) is a 2-D interactive graphics package which allows a user to describe a limited set of linkage components (to be described), combine these components into linkages, change the linkage and/or interactively drive the linkage through its inherent motion cycle (or parts thereof) to visualize its mobility. The linkage components selected yield a simplified kinematics treatment to be described.

The thrust of the work has primarily been to define and implement a software system for accomplishing the above capabilities, to investigate the usefulness of this approach, to define problem areas, and to identify potentially fruitful areas for future research.

In order to tackle the visualization goal of this project without getting bogged down in the details of analysis, some limitations are set on the approach to topological definition and kinematic analysis.

An earlier-presented analysis technique described by Shu and Radcliffe (2) in 1978 forms the basis of the approach used in this project. In this analysis method, a set of kinematic "elements" is defined from which linkages can be formed.

Associated with each kinematic element or component is a set of FORTRAN subroutines. The user writes a main program which

1. Sets up geometry, initial conditions, etc.

2. Calls kinematic subroutines in required sequence to solve for all motion results.

3. Handles output (tables, graphics, etc.).

In other words the user handles I/O in steps 1 and 3, and topology is addressed in step 2. As spartan as this might sound, some very complex linkages can be handled with this technique. In fact, as recently as mid-1984 Smith and Ye (45) reported the creation of an input language called SIMPLA, which supports the above method of Shu and Radcliffe.

Kinematic analysis in LKSP is based on a modification of Shu and Radcliffe's procedure. Since LKSP will use interactive graphics for input and output, steps 1 and 3 are covered.

To study the kinematic mobility of planar linkages, a "live" kinematic model, i.e. one that can easily be interactively changed and repeatedly run, is required. It seems desirable to incorporate a simplification of the allowed kinematic elements which would yield a model requiring no topological analysis and a well defined sequential computational procedure regardless of the way the basic components are combined.

To these ends, a set of kinematic components is defined (see Figure 1-2) such that

1. Each element is either a driver or a nondriver.

    a. Drivers may be rotating or translating.

    b. Nondrivers have one input point and one output point. Nondrivers include a dyad-1, dyad-2, rotating guide, oscillating slider and a slider. There is a choice of two possible output points for the dyad-2.

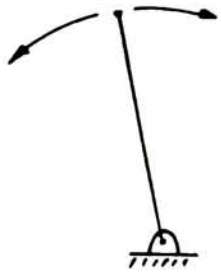
2. Each component is grounded, thereby eliminating topological uncertainty. As elements are connected a loop is formed and a computational sequence for the kinematics equations is defined.

3. One driver and any number of nondrivers are allowed to be connected together.

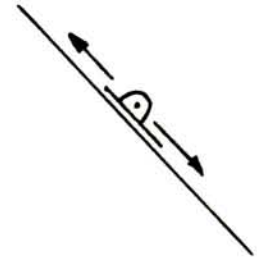
Figure 1-3 shows two examples of linkages formed from the above described linkage components. Even though the linkage elements are simple, many useful linkages can be created.

In order to perform the kinematic analysis within LKSP, assembly and motion algorithms have been developed for each component type. These are described in Tables 1-2 through 1-8. Table 1-1 explains a set of supporting trigonometric functions and coordinate transform procedures. A graphical illustration of each component is shown in Figures 1-4 through 1 10.

Drivers.

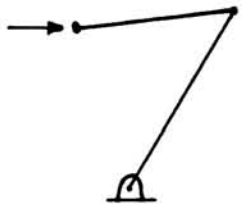


Rotational Driver.

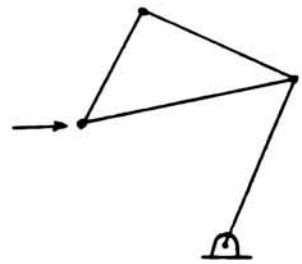


Translational Driver.

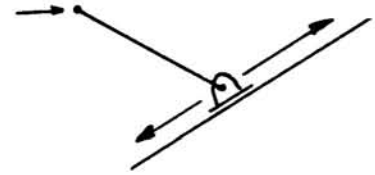
Nondrivers.



Dyad-1.

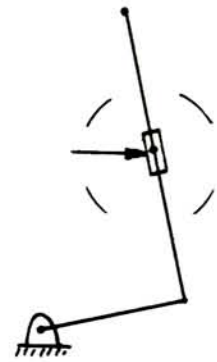


Dyad-2.



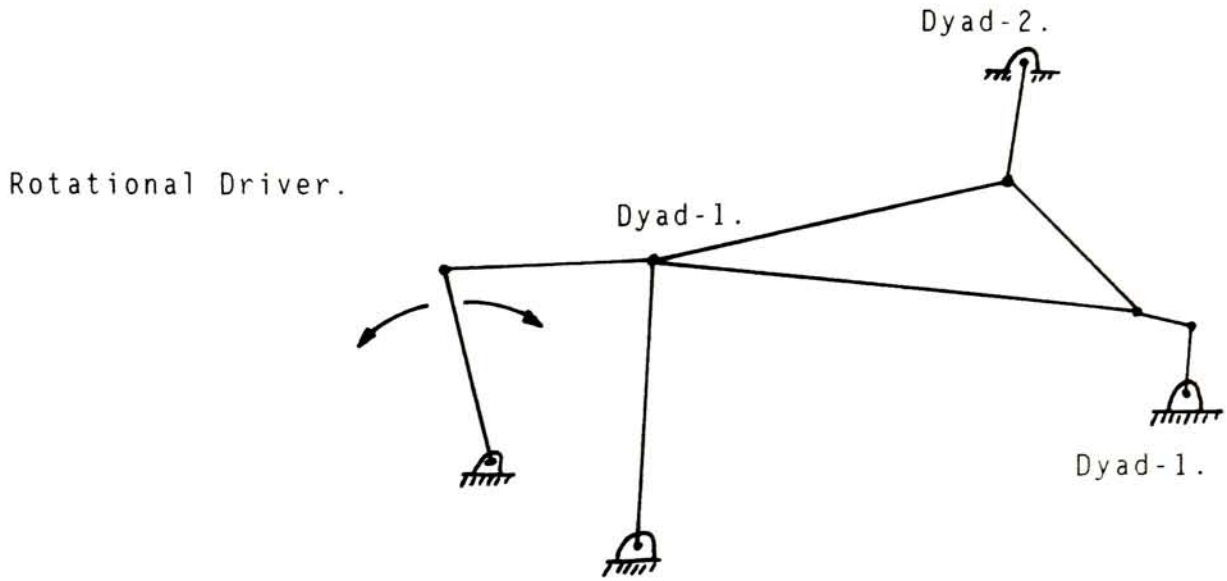
Slider.

Rotating Guide.

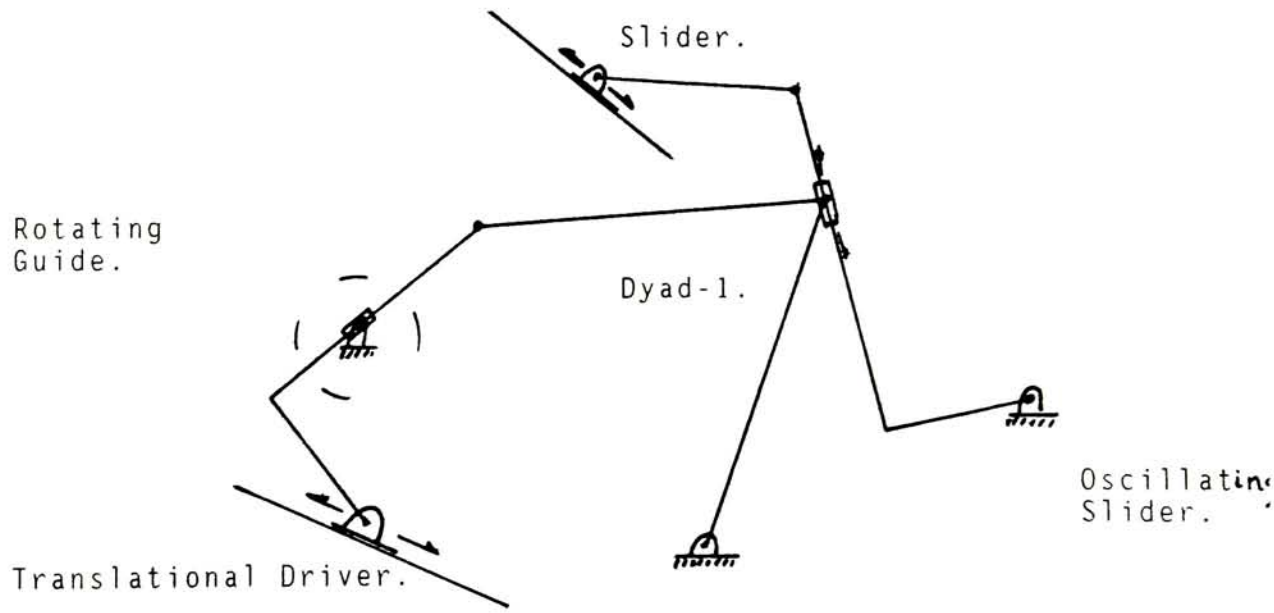


Oscillating Slider.

Figure 1-2 Proposed Linkage Components, Drivers and Nondrivers.



3-Loop, Multiple 4-Bar Linkage.



4-Loop, Combination Linkage.

Figure 1-3 Example Linkages.

Table 1-1 Trigonometric Functions and Coordinate Transforms Used in Kinematics Algorithms.

Trigonometric Functions:

```
function Length( xlin, ylin, x2in, y2in : real ):real;
                        { How long is line from global coordinates }
                        { xlin, ylin to x2in, y2in?                }

function GetAngleH( xlin, ylin, x2in, y2in : real ):real;
                        { What is the angle, in radians,    }
                        { formed by the line 2-1-h, i.e.    }
                        { from x2in, y2in to xlin, ylin     }
                        { to h, horizontal? Result = 0..2pi.}

function GetAngleT( xlin, ylin, x2in, y2in : real ):real;
                        { What is the angle, in radians,    }
                        { formed by the line 2-1-h, i.e.    }
                        { from x2in, y2in to xlin, ylin     }
                        { to h, horizontal. Differs from    }
                        { GetAngleH because result is       }
                        { resolved to +/- 0..pi/2.          }

function AddAngles( anglelin, angle2in : real ):real;
                        { What is the sum of 2 angles?  }
                        { Result is resolved to 0..2pi. }

function GetArcSin( ain, hin : real ):real;
                        { What is the angle formed by }
                        { the sides ain & hin,        }
                        { where ain is the opposite   }
                        { side & hin is the hyp. &    }
                        { whose Sin is the ratio of   }
                        { ain/hin? Result = 0..2pi.   }

function GetInteriorAngle( ain, bin, cin : real ):real;
                        { What is the interior angle   }
                        { in radians formed by line    }
                        { ain, bin & opposite to cin?  }
                        { Result = 0..pi.              }
```

Coordinate Transforms:

```
procedure Transform( theta, tx, ty, x, y, xTR, yTR );
                        { Transform global coordinates x, y into   }
                        { xTR, yTR , slider coordinates, using CTM.}
                        { CTM is set to rotate angle theta &       }
                        { translate tx, ty.                        }

procedure Back_Transform( theta, tx, ty, x, y, xTR, yTR );
                        { Reverse transform, xTR, yTR back into   }
                        { x,y using above CTM.                    }
```

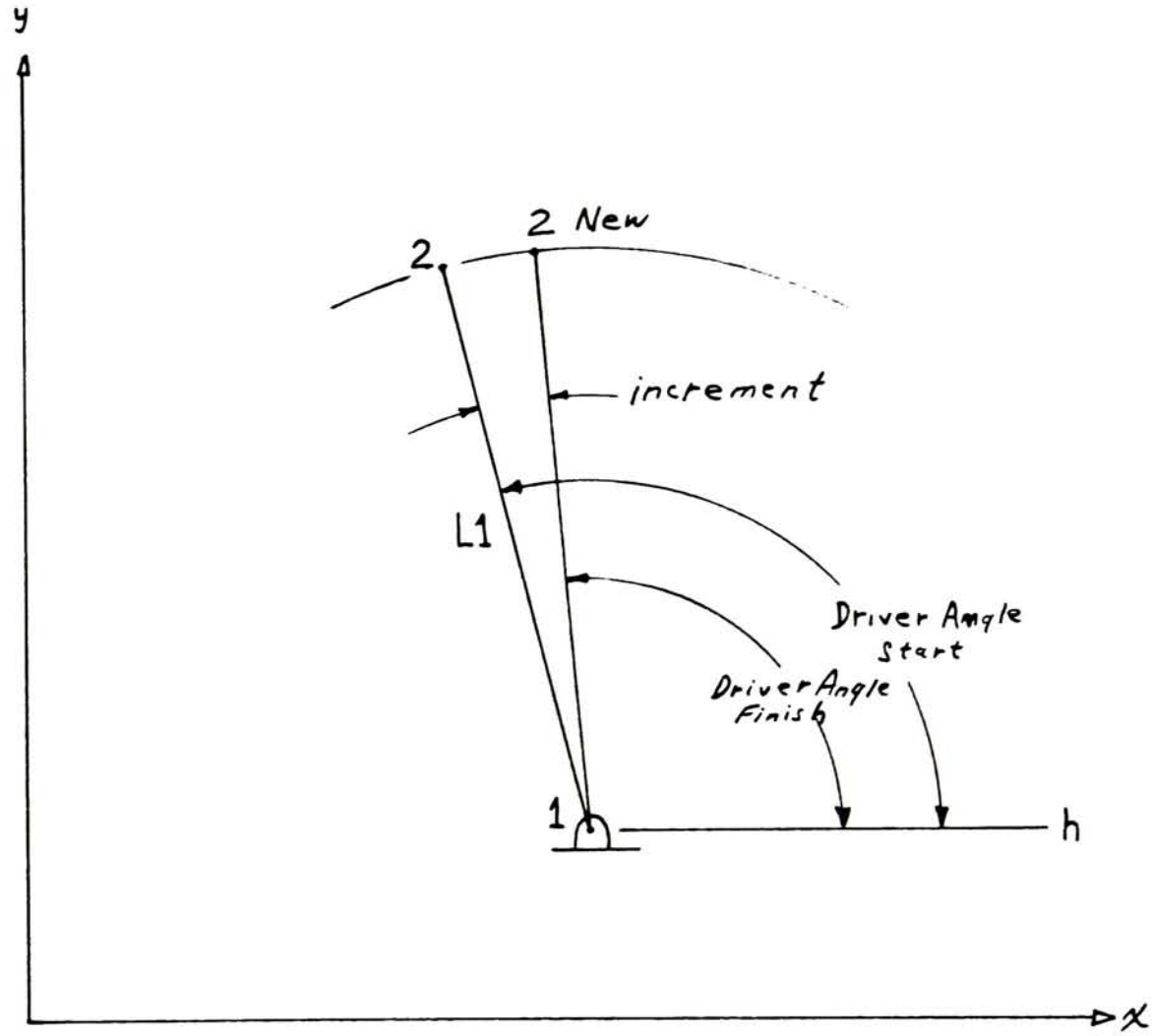


Figure 1-4 Rotating Driver.

Table 1-2 Rotating_Driver Algorithms.
(See Figure 1-4)

Initial Assembly:

```
x1, y1, x2, y2 are user input.
pt 1 is base, pt 2 is drive point.

Compute length of driver & Driver_Angle:
        L1 := Length( x1, y1, x2, y2 );
        Driver_Angle := GetAngleH( x1, y1, x2, y2 );
```

Motion:

```
Motion is incremental in steps of size motion_increment
(a rotational step size, typically 2 degrees).

Point 2 coordinates, L1 & motion_increment are constant.

Set-up motion variables:

        Current_Angle := Driver_Angle;

        Current_x2 := x2;

        Current_y2 := y2;


Implement motion:

     if( Current_Angle = Target_Angle ) then Motion_Complete
     else begin

        Current_Angle := Current_Angle + motion_increment;

        New_x2 := Current_x2 + L1 * cos( Current_Angle );

        New_y2 := Current_y2 + L1 * sin( Current_Angle );

        Current_x2 := New_x2;

        Current_y2 := New_y2;

     end;

     Driver_Angle := Current_Angle;
```

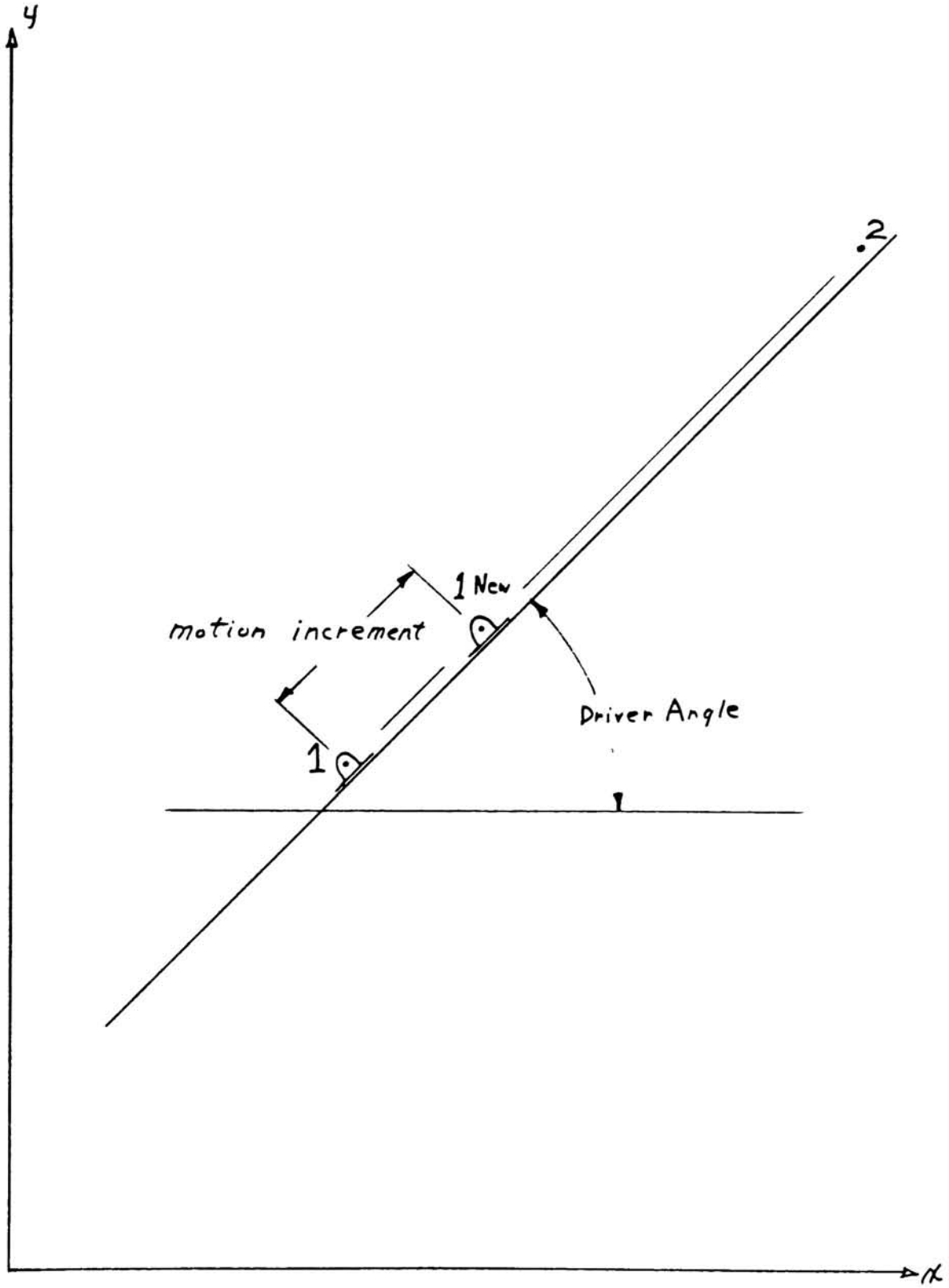


Figure 1-5 Translating Driver.

Table 1-3 Translating_Driver Algorithms.
(See Figure 1-5)

```
Initial Assembly:

   x1, y1, x2, y2 are input.
   pt 1 is base, pt 2 defines angle.

   Compute angle of driver.
           Driver_Angle := GetAngleT( x1, y1, x2, y2 );


Motion:

   Motion is incremental in steps of size motion_increment
   (a translational step size, typically 0.10 inch).

      Driver_Angle & motion_increment are constant.

      Set-up motion variables:

           Current_x1 := x1;

           Current_y1 := y1;

           Current_Position & Target_Position formed
           from x & y components.

      Implement motion:

           if( Current_Position = Target_Position ) then Motion_Complete
           else begin

                   New_x1 := Current_x1 + motion_increment * cos( Driver_Angle );

                   New_y1 := Current_y1 + motion_increment * sin( Driver_Angle );

                   Current_x1 := New_x1;

                   Current_y1 := New_y1;
           end;
```

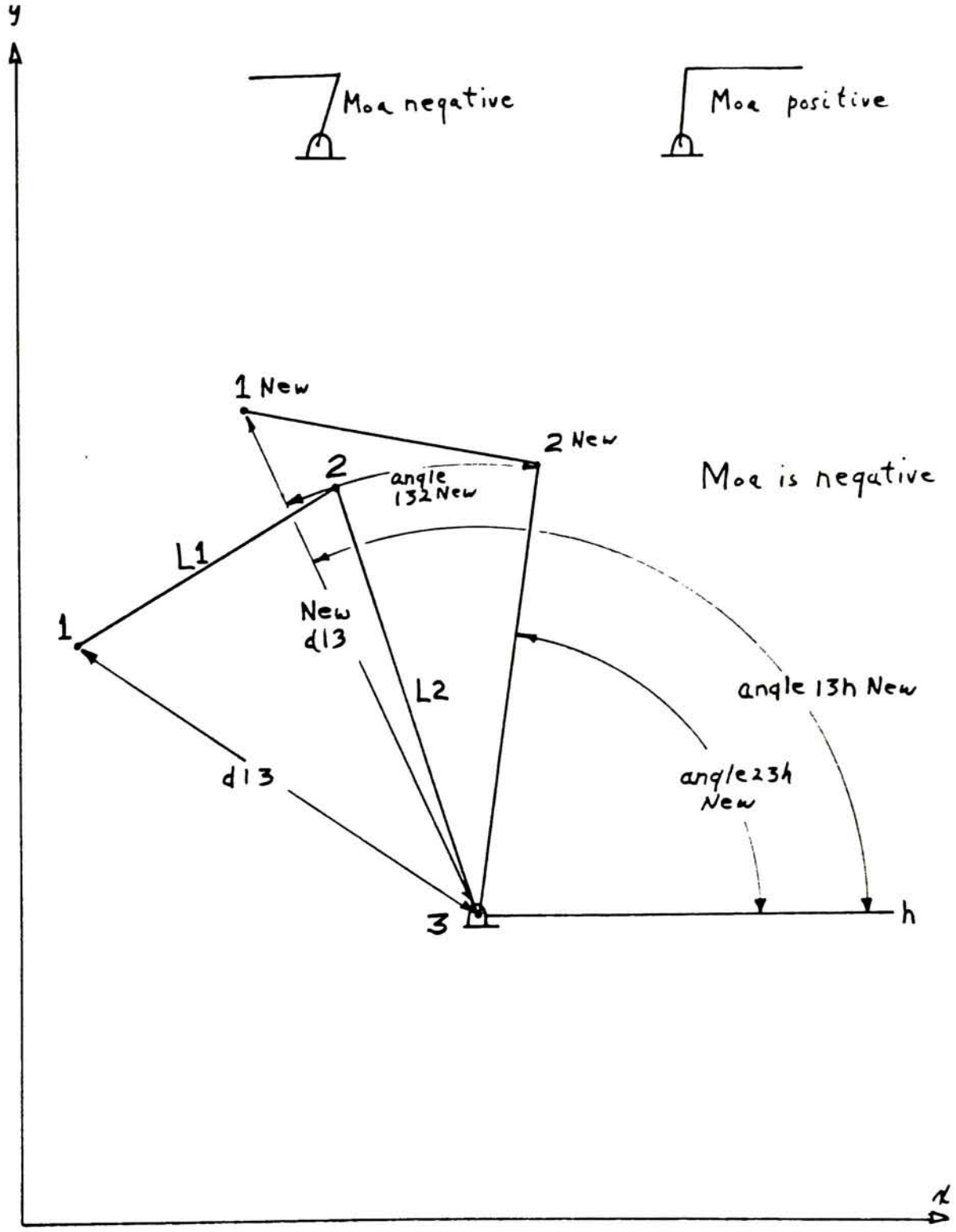


Figure 1-6 Dyad-1.

Table 1-4 Dyad-1 Algorithms.
(See Figure 1-6)

```
Initial Assembly:

   x1, y1, x2, y2, x3, y3 are input.

   Compute link & characteristic lengths:

       link lengths:
           L1 := Length( x1, y1, x2, y2 );
           L2 := Length( x2, y2, x3, y3 );

       characteristic lengths:
           longest  := L1 + L2;       { longest possible length  }
           shortest := abs( L1   l2); { shortest possible length }

   Determine Mode of Assembly,  Moa:

       if angle23h = AddAngles ( angle13h, angle132 )
           then Moa is positive else Moa is negative;

Motion:

   New_x1  &  New_y1  are input during motion.
   x3 & y3 are fixed.  L1 & L2 are fixed.
   Moa is fixed.

   Check for binding:

           New_d13 := Length( New_x1, New_y1, x3, y3 );

           if(  ( New_d13 > longest  )  or
                ( New_d13 < shortest )    )  then
                   binding := true, stop all motion
           else binding := false so begin

   Calculate new position, point 2, ( New_x2, New_y2 ):

           angle13h := GetAngleH( x3, y3, New_x1, New_y1 );
           angle132 := GetInteriorAngle( New_d13, l2, l1 );

           if ( Moa = positive )
             then angle23h := AddAngles ( angle13h, angle132 )
           else if ( Moa = negative )
             then angle23h := AddAngles ( angle13h,   angle132 );

           New_x2 := x3 + L2 * cos( angle23h );
           New_y2 := y3 + L2 * sin( angle23h );
```

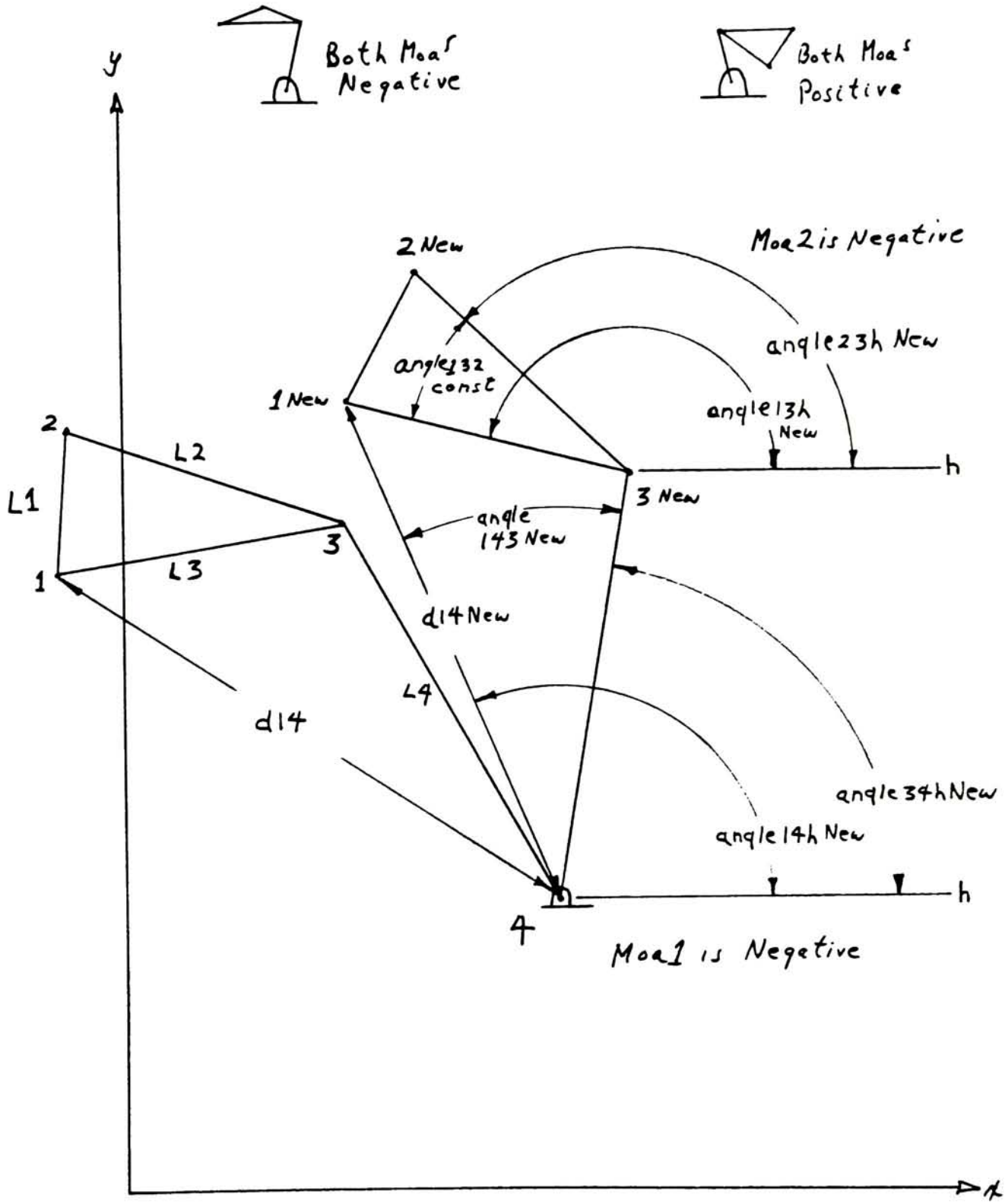


Figure 1-7 Dyad-2.

Table 1-5 Dyad-2 Algorithms.
(See Figure 1-7)

```
Initial Assembly:

   x1, y1, x2, y2, x3, y3, x4, y4 are input.
   Compute link lengths L1, L2, L3 & L4 from input x, y set.
   Compute characteristic lengths:
           longest  := L3 + L4;       { longest possible length  }
           shortest := abs( L3   L4); { shortest possible length }
   Compute characteristic angle for panel formed by points 1,2,3:
           angle132 := GetInteriorAngle( L3, L2, L1 );

   Determine Modes of Assembly,  Moa1, Moa2:
           if angle34h = AddAngles ( angle14h, angle143 )
           then Moa1 is positive else Moa1 is negative;

           if angle23h = AddAngles ( angle13h, angle132 )
           then Moa2 is positive else Moa2 is negative;

Motion:

   New_x1 & New_y1 are input during motion.
   x4 & y4, L1 through L4, Moa1, Moa2 & angle132 are fixed.

   Check for binding:
           New_d14 := Length( New_x1, New_y1, x4, y4 );

           if(  ( New_d14 > longest  ) or
                ( New_d14 < shortest )   ) then
                   binding := true, stop all motion
           else binding := false so begin

   Calculate new position, point 3 ( same as Dyad-1 ):

           angle14h := GetAngleH( x4, y4, New_x1, New_y1 );
           angle143 := GetInteriorAngle( New_d14, L3, L4 );

           if ( Moa1 = positive )
             then angle34h = AddAngles ( angle14h, angle143 )
           else if ( Moa1 = negative )
             then angle34h = AddAngles ( angle14h,   angle143 );

           New_x3 := x4 + L4 * cos( angle34h );
           New_y3 := y4 + L4 * sin( angle34h );

   Calculate new position of point 2 ( New_x2, New_y2 ):

           angle13h := GetAngleH( New_x3, New_y3, New_x1, New_y1 );
           if ( Moa2 = positive )
             then angle23h := AddAngles ( angle13h, angle132 )
           else if ( Moa2 = negative )
             then angle23h := AddAngles ( angle13h, - angle132 );

           New_x2 := NewX3 + ( L2 * cos( angle23h )  );
           New_y2 := NewY3 + ( L2 * sin( angle23h )  );
```

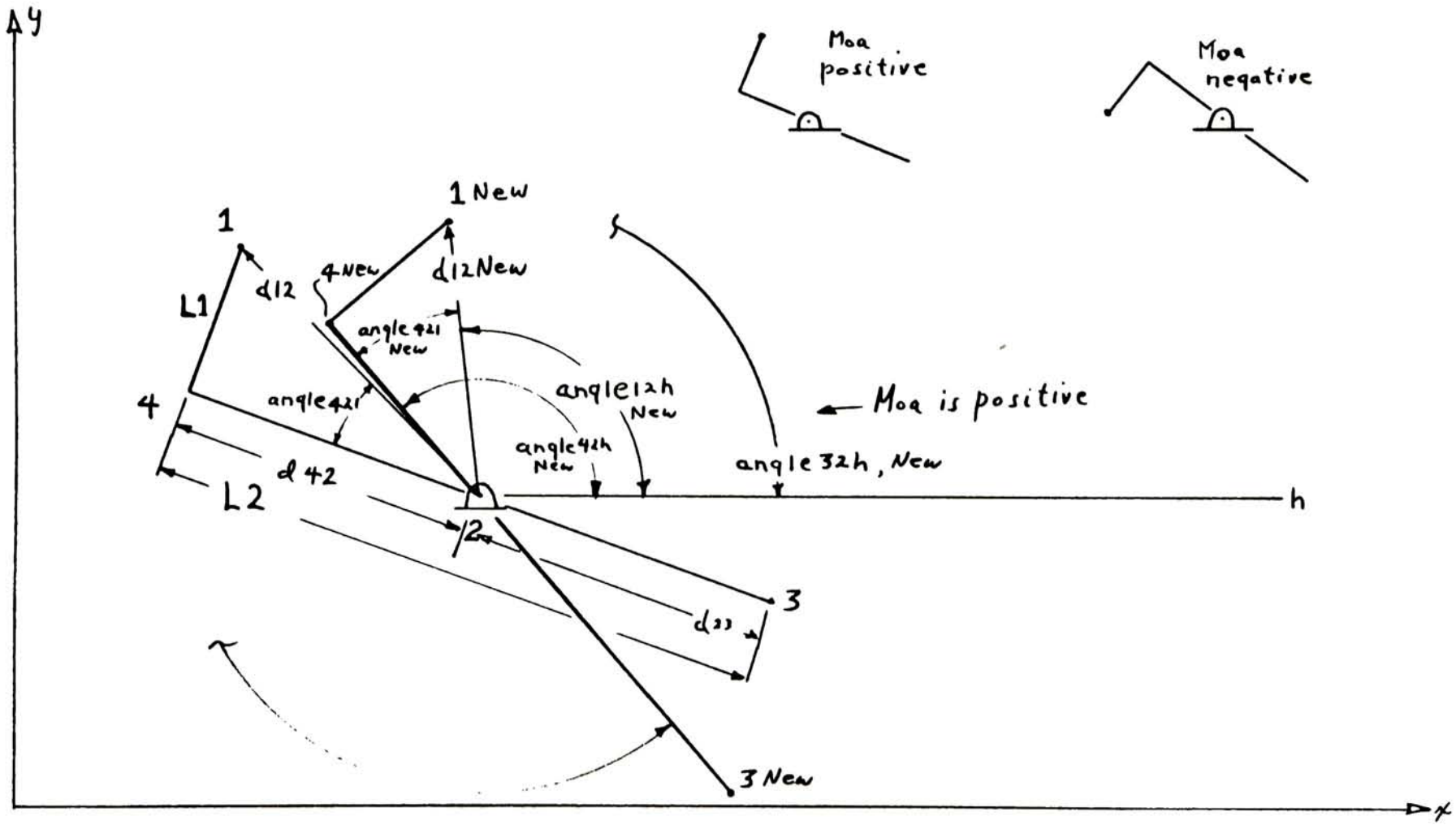


Figure 1-8 Rotating Guide.

Table 1-6 Rotating_Guide Algorithms.
(See Figure 1-8)

```
Initial Assembly:

   x1, y1, x2, y2, x3, y3 are input.

   Compute d12, d23 & angles:
           d12 := Length( x1, y1, x2, y2 );
           d23 := Length( x2, y2, x3, y3 );
           angle12h := GetAngleH( x2, y2, x1, y1 );
           angle32h := GetAngleH( x2, y2, x3, y3 );

   Find Point 4:
           angle42h := angle32h + 180 degrees;
           L1 := d12 * sin( angle42h   angle12h );
           angle421 := GetArcSin( l1, d12 );
           d42 := d12 * cos( angle421 );
           x4 := d42 * cos( angle42h ) + x2;
           y4 := d42 * sin( angle42h ) + y2;

   Compute Characteristic Lengths:
           L2 := d23 + d42;
           shortest := L1;
           longest  := L2;

   Determine Mode of Assembly, Moa:
           if ( angle42h := AddAngles ( angle12h, angle421 )
           then Moa = positive else Moa = negative;

Motion:

   New_x1  &  New_y1  are input during motion.

   x2 & y2 are fixed. L1 & L2 are fixed.
   Moa ( the mode of assembly ) is fixed.

   Check for binding:

           New_d12 := Length( New_X1, New_Y1, x2, y2 );
           if(  ( New_d12 > longest ) or ( New_d12 < shortest )    )
                   then binding := true, stop all motion,
           else binding := false so begin

   Calculate new positions, points 4 & 3:

           angle12h := GetAngleH( x2, y2, New_X1, New_Y1 );
           angle421 := GetArcSin( L1, New_d12 );
           if ( Moa = positive )
              then angle42h := AddAngles ( angle12h, angle421 )
           else if ( Moa = negative )
              then angle42h := AddAngles ( angle12h,   angle421 );

           angle32h := angle42h + 180.0 degrees;
           d42 := d12 * cos( angle421 );
           New_x4 := d42 * cos( angle42h ) + x2;
           New_y4 := d42 * sin( angle42h ) + y2;
           d23 := l2   d42;
           New_x3 := d23 * cos( angle32h ) + x2;
           New_y3 := d23 * sin( angle32h ) + y2;
```

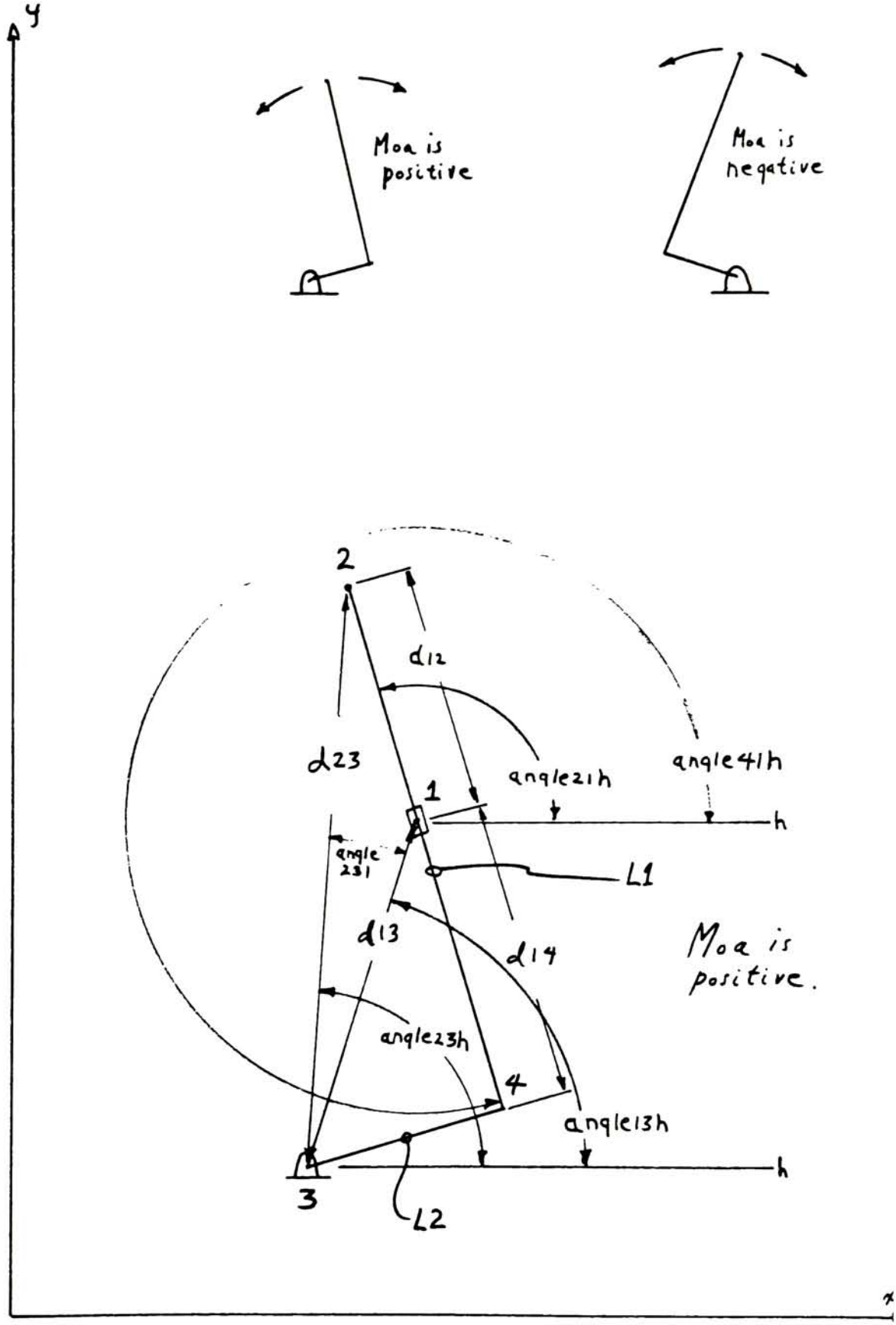


Figure 1-9 Oscillating Slider.

Table 1-7 Oscillating_Slider Algorithms.
(See Figure 1-9)

```
Initial Assembly:

   x1, y1, x2, y2, x3, y3 are input.

   Compute d12, d13, d23 & angles:
           d12 := Length( x1, y1, x2, y2 );
           d13 := Length( x1, y1, x3, y3 );
           d23 := Length( x2, y2, x3, y3 );
           angle13h := GetAngleH( x3, y3, x1, y1 );
           angle21h := GetAngleH( x1, y1, x2, y2 );
           angle23h := GetAngleH( x3, y3, x2, y2 );
           angle41h := angle21h + 180 degrees;
           angle321 := GetInteriorAngle( d23, d12, d13 );
           angle231 := GetInteriorAngle( d23, d13, d12 );

   Find characteristic lengths, L1, L2 & angle:
           L2 := d23 * sin( angle321 );
           L1 := d23 * cos( angle321 );
           angle234 := GetInteriorAngle( d23, l2, l1 );
           longest := d23; shortest := L2;

   Find Point 4:
           d41 := L1   d12;
           x4 := x1 + ( d41 * cos( angle41h )  );
           y4 := y1 + ( d41 * sin( angle41h )  );

   Determine Mode of Assembly:
           if ( angle23h = angle13h + angle231 )
           then Moa = positive else Moa = negative;

Motion:
   New_x1 & New_y1 are input during motion.
   x3 & y3 are fixed. L1, L2 & angle234 are fixed.
   Moa ( the mode of assembly ) is fixed.

   Check for binding:
           New_d13 := Length( New_x1, New_y1, x3, y3 );
           if(  ( New_d13 > longest ) or ( d13 < shortest )  )
           then binding := true else binding := false so begin

   Calculate new position, points 4 & 2:
           d41 := sqrt(  sqr( New_d13 )   sqr( L2 )  );
           d12 := L24   d41;
           angle13h := GetAngleH( x3, y3, New_x1, New_y1 );
           angle231 := GetInteriorAngle( d23, New_d13, d12 );
           if ( Moa = positive ) then begin
                   angle23h := AddAngles ( angle13h, angle231 );
                   angle43h := AddAngles ( angle23h,   angle234 )
           end else if ( Moa = negative ) then begin
                   angle23h := AddAngles ( angle13h,   angle231 );
                   angle43h := AddAngles ( angle23h, angle234 )
           end;

           New_x2 := x3 + ( d23 * cos( angle23h )  );
           New_y2 := y3 + ( d23 * sin( angle23h )  );
           New_x4 := x3 + ( L2 *  cos( angle43h )  );
           New_y4 := y3 + ( L2 *  sin( angle43h )  );
```

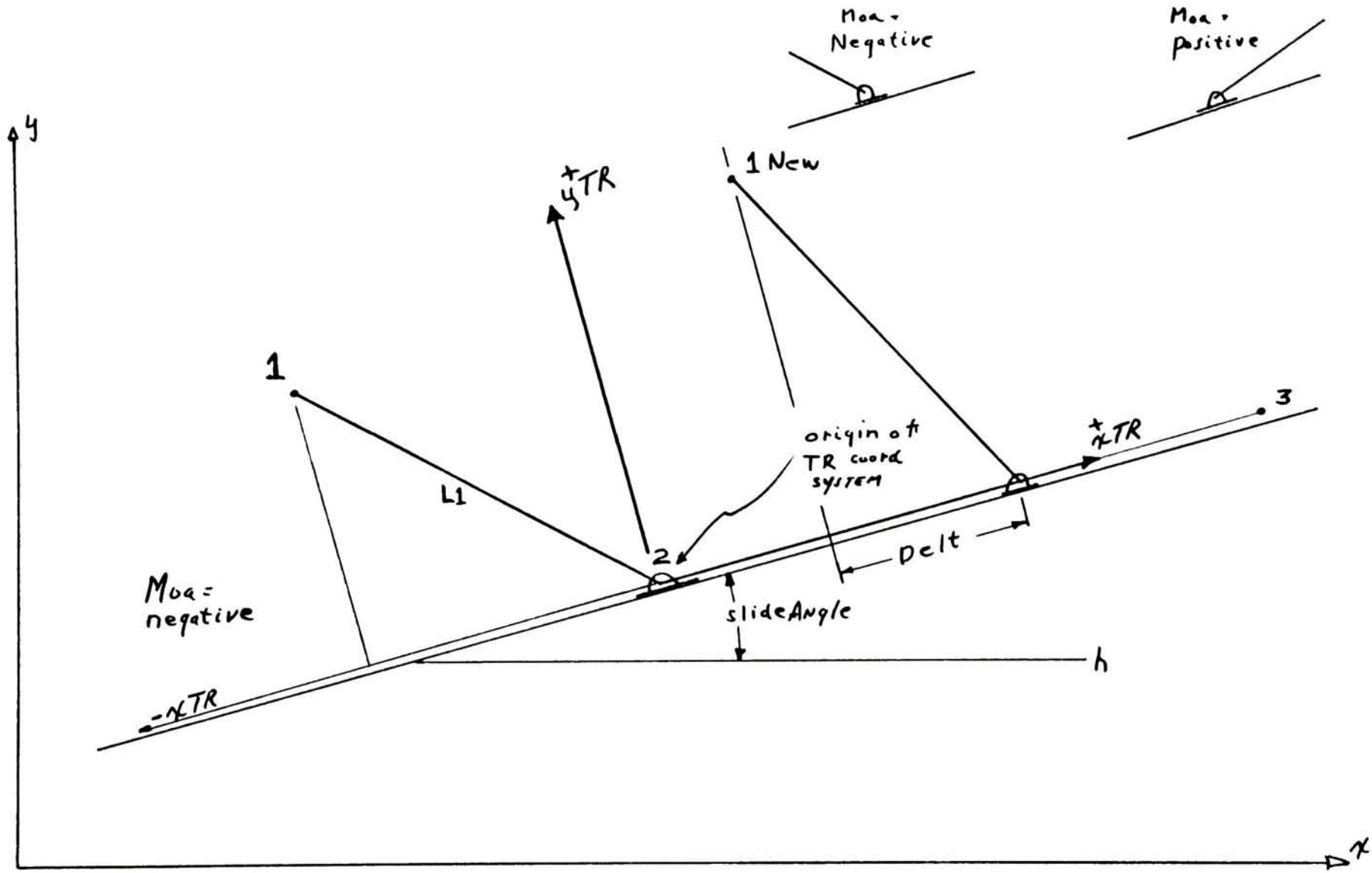


Figure 1-10 Slider.

Table 1-8 Slider Algorithms.
(See Figure 1-10)

```
Initial Assembly:

   x1, y1, x2, y2, x3, y3 are input.
   pt. 3 defines slide angle & pt. 2 defines starting point
   which is used as the origin of the slider coordinate system
   (labeled xTR, yTR).
           startx := x2;
           starty := y2;

   Compute characteristic length & angle:
           L1 := Length( x1, y1, x2, y2 );
           longest := L1; shortest := - L1; { longest= - shortest }
           slideAngle := GetAngleT( x2, y2, x3, y3 );  { angle21h }


   Determine Mode of Assembly:
           Transform( slideAngle,
                      startx, starty,       { transform x1, y1  }
                      x1, y1,               { into slider coord }
                      x1TR, y1TR );         { x1TR, y1TR        }
           if( x1TR >= 0.0 )                { Moa determined by }
             then Moa := positive           { relative position }
             else Moa := negative;          { of pts 1 & 2      }
                                            { see Figure 1-10   }

Motion:

   New_x1  &  New_y1  are input during motion.
   startx, starty, L1 & slideAngle are fixed.
   Moa ( the mode of assembly ) is fixed.

   Check for binding:
           Transform( slideAngle,
                      startx, starty,  { transform New_x1, New_y1 }
                      New_x1, New_y1,  { into slider coord        }
                      New_x1TR,        { New_x1TR, New_y1TR       }
                      New_y1TR );

           if(     ( New_y1TR > longest  ) or
                   ( New_y1TR < shortest )  )
              then binding := true else binding := false so begin

   Find new position pt. 2 (i.e. New_x2, New_y2):

           Delt := sqrt(  sqr( L1 )   sqr( New_y1TR )  );
           if ( Moa = positive ) then Delt :=   abs( Delt )
           else if ( Moa = negative ) then Delt := abs( Delt );
           New_x2TR := New_x1TR + Delt;
           New_y2TR := 0;

           Back_Transform( slideAngle,
                      startx, starty,  { convert slider coord,     }
                      New_x2, New_y2,  { New_x2TR, New_y2TR, back }
                      New_x2TR,        { into global,  New_x2,     }
                      New_y2TR );      { New_y2                    }
```

2. Project Description.

2. Project Description.

As mentioned in section 1.1, the main thrust of this project has been the development of a prototype interactive graphics (CAD) system aimed at visualizing the motion and mobility of linkage-type, planer mechanisms.

The program is called the Linkage Kinematics Sketchpad (LKSP). It is a 2-D color graphics package which allows the user to build linkages from the previously described set of linkage components and interactively drive the linkage on the screen to visualize its mobility.

In this section, LKSP is described. Section 2.1 illustrates what the package will do from a user's perspective, i.e. the functions which can be performed (2.1.1), the functions which cannot be performed (2.1.2), examples of input and output with screen images (2.1.3), and a description of the file structures (2.1.4).

Section 2.2 describes the software and hardware. A system organizational chart (2.2.1) is presented which pictorially describes software module hierarchy, inter-module calling sequences and communications. Next, a graphical description of information flow among the software modules is presented as a system data flow chart (2.2.2). A brief description of hardware used for this project is presented (2.2.3) followed by a comment on implementation tools (2.2.4), i.e. language and other software tools.

2.1 Functional Description

This section describes what LKSP does from a user's perspective. Before getting into specifics, it is necessary to state some preliminary rules regarding I/A. Easy input was one goal of this work and to that end, all user input is cursor oriented (e.g. a pick) as opposed to keyboard. Also, the available screen area has been divided into three areas, dialog, menu, and working display. The menu space is devoted to allowing user interaction (i.e. picking functions and sub-functions). The graphics display area or "working space" is devoted to display of linkages. Certain user picking functions are also allowed here. The dialog area is devoted to prompts and messages (errors warnings, help, status, etc.).

The following sections show the functions which can be performed with LKSP (2.1.1) as well as the functions which cannot be performed (2.1.2), give examples of input and output (2.1.3), and a description of the file structures (2.1.4).

2.1.1 Functions Performed

The following is a description of the functions which are performed by LKSP. These are categorized into 6 main areas:

1. Gridding.

2. Creating the linkage.

3. Deleting linkage components.

4. Moving the linkage.

5. Tracing linkage motion.

6. Completion.

Gridding functions allow selection of two (fine and coarse) x-y grid patterns to be displayed in the working space. These grids allow reasonably close (although approximate) placement and sizing of components without the need for keyboard input. The grid can be turned "on" or "off" by the user whenever menu picks are allowed.

Under the general heading of linkage creation, a number of functions are included. First (and foremost) users are able to select component types and add them to the end of the linkage under construction (i.e. place the component in the display area). During the addition process, the user graphically picks locations for the component points. In the specific case of driver type components (there are two of these), the user has the capability of replacing one variety with the other.

Also included under the general heading of linkage creation (more correctly "creation" alone) are two types of "interference panels" which users can select, and place in the display area. These "I-panels" include a three-sided and a four-sided variety. Interference panels are intended to simulate space constraints on the mobility of the linkage which the user wishes to investigate.

A user can "delete" any component in the overall linkage except drivers. A driver can only be replaced with the other type of driver. Deletion of any "internal" component (i.e. not the most recently described or last in the chain) has some interesting implications which will be discussed under I/O (2.1.3).

Moving the linkage calls on the kinematic analysis capability of LKSP. Users can "move" the overall linkage or "drive" the linkage through its inherent motion cycle (or parts thereof) and observe its mobility. The user is required to "pick" a

position in the display area near the driver. The linkage moves in a direction dictated by the location of the pick relative to the current position of the driver.

If motion becomes impossible because of the configuration of the linkage (i.e. the linkage gets "bound up"), motion stops until the next move request (e.g. in the opposite direction). A blinking message is displayed on the screen stating that the assembly is binding. Linkages assembled according to user input have a certain inherent "mode of assembly" (see Chapter 1). The mode of assembly is preserved in LKSP (i.e. the mode of assembly never changes).

The occurrence of interference during motion is visible due to use of colors for interference panels different from the linkage colors, but the linkage is allowed to pass through the interference panel.

The user can request that all points on the linkage which move be traced during motion. Also these points can be "turned-off". In addition, a "clean-up" feature is available to eliminate old traces when they have outlived their usefulness. The completion function allows the user to end the program.

2.1.2 Limitations and Restrictions

The following is a summary of the functions which are not performed by LKSP. Under the general heading of analysis, there is a class of mechanisms which are implicitly excluded from consideration by LKSP. These include any mechanisms with open chains (Figure 1 1) or components with more than one input point. A few examples of the latter are illustrated in Figure 2-1. In addition, LKSP is not intended to do synthesis, three-dimensional kinematic analysis, kinetics, gears or cams, elastic linkages or external stops.

Other limitations include

1. only one driver is allowed.

2. only one deletion mode (e.g. the user is not able to disconnect parts of a linkage for subsequent re-use).

3. only "pick" input (e.g. user cannot directly specify exact lengths of components).

4. linkage shows crossing of interferences only by color change instead of stopping.

5. current implementation excludes component modification (except deletion).

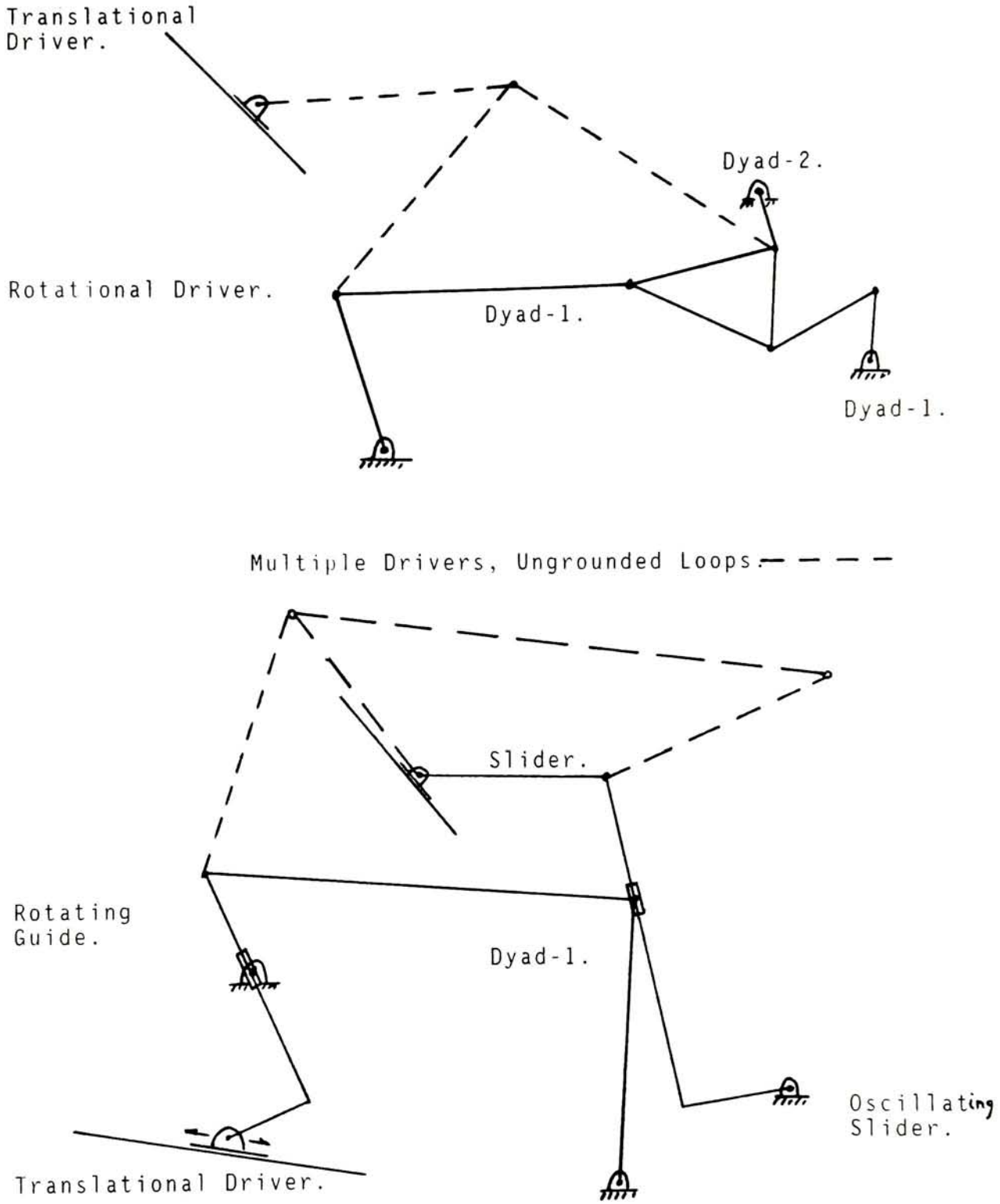


Figure 2-1 Linkages Which Cannot Be Described Within LKSP.

6. current implementation excludes modification of tracer points.

7. current implementation excludes creation of a save file with geometry information.

2.1.3 User Inputs and Outputs

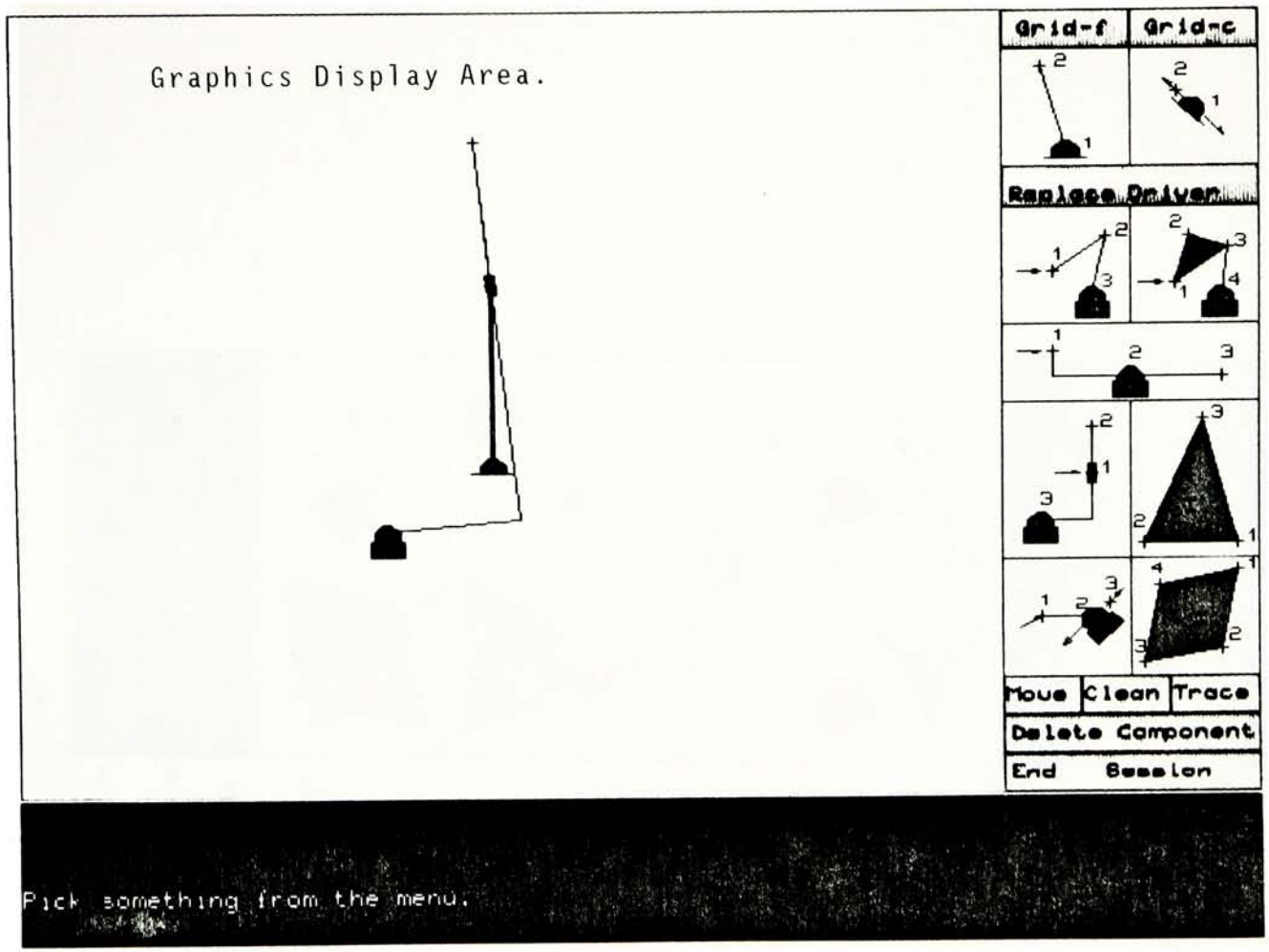
In this section user inputs and outputs are described. As previously mentioned, user input is by cursor (e.g. a menu pick) and the available screen is divided into three main areas: dialog, menu and working display. The menu space is devoted to allowing user interaction (i.e. picking functions and subfunctions). The display area is devoted to display of graphic images of the linkage and certain user picking functions. The dialog area is devoted to prompts and messages. Figure 2-2 illustrates the screen layout.

The menu has been functionally laid out from top to bottom as shown in Figure 2-3. An attempt was made to gather functionally similar items on the menu. The icons are obvious enough for most users. Figure 2-3 was created from an actual, full size paper copy of the color screen image. Original paper copies of color screen images were created with a Tektronix 4695 ink-jet raster plotter. The copies of these originals, which appear as illustrations in this report, have slightly different shades of color than the originals.

At the start of a session, a user can pick either driver. but the user cannot pick one of the other components until a driver is picked. Error messages and prompts appear to guide the user if a mistake is made. "Replace Driver" cannot be picked until a driver exists after which the driver can be replaced at any point where menu picks are allowed.

Interference panels can be selected at any point except (obviously) during selection of another component. Also a grid (fine or coarse) can be requested at any time when menu picks are allowed (a prompt appears stating "Pick something from the menu") as can "Trace", " Clean", "Move" and "End Session." Legal menu selections are acknowledged by displaying the selected function with an accenting color or blinking background color until the function is complete. In the remainder of this section, examples of accessing various menu items will be presented.

At the start of a session the menu is displayed and a welcome message appears in the graphics display area. The dialog area appears (blue background, can hold five lines). The cursor appears along with a prompt in the dialog area telling the user what can be done next. In Figure 2-2 the message is telling the user to "Pick something from the menu".



Menu Area.

Dialog Area.

Figure 2-2 Screen Layout.

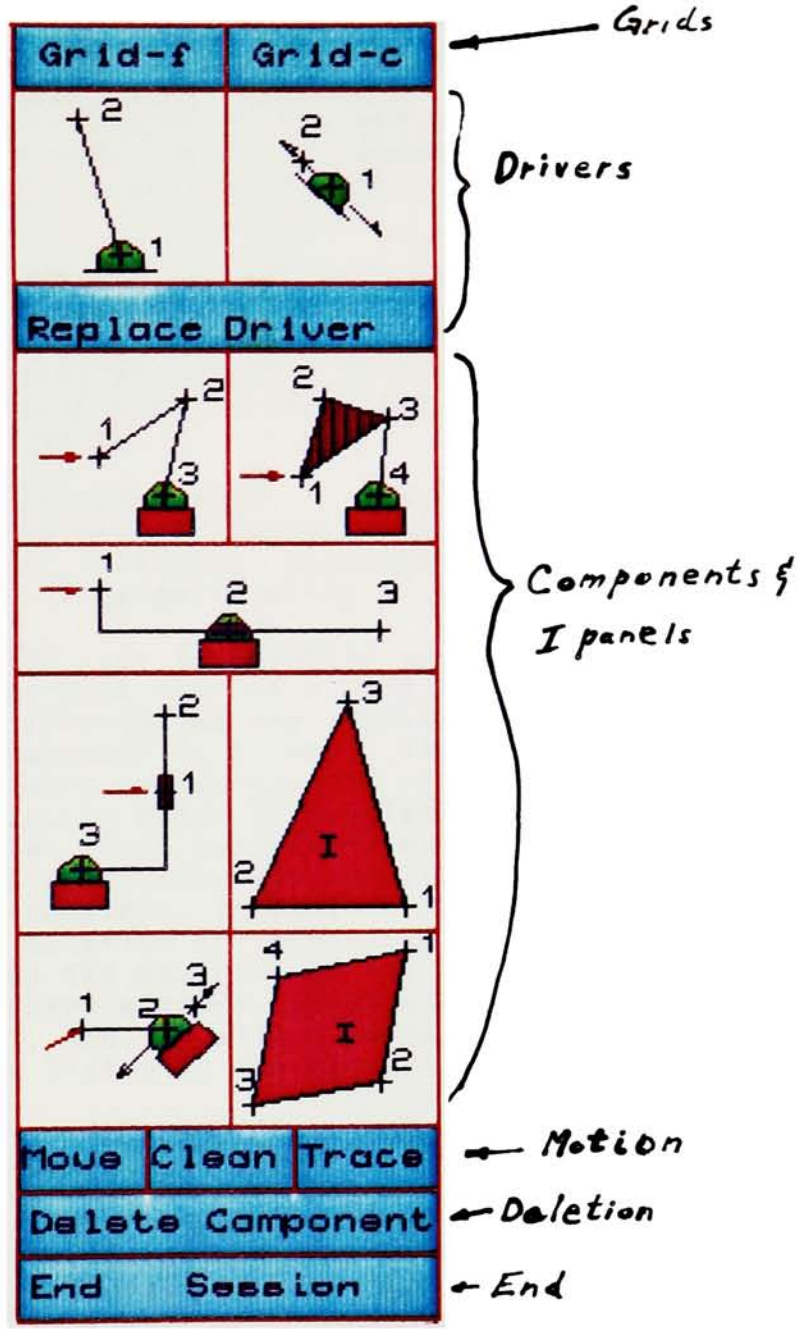


Figure 2-3 Menu Layout.

If the user picks a rotating driver, a prompt appears telling the user how many points to pick and the order in which the points are expected. The point numbers correspond to the numbers in the icons so the user always knows their topological meaning.

After point 1 is picked (the base in this case), a base appears in the graphics display area centered on the pick location. After the second pick (point 2, the end of the rotating driver), the entire rotating driver appears. Figure 2-4 shows a rotating driver (in green) with other components and two interference panels.

During replacement of the two driver types, the previously picked driver-connection point, or point of attachment to the other linkage components (whether or not they exist), cannot change. A single pick is required in the case of replacement, to define the angle of a new translating driver or the base of a new rotating driver.

Selection of interference panels is simply a matter of picking the three or four points outlining the panel. The panel appears after all points are selected. Rubberbanding is used during point selection to allow positioning of edges.

During deletion, the user is prompted to pick the component to be deleted. The pick must be within the base of the component thereby eliminating any potential confusion regarding which component is picked. Since each component has a base, this is a very convenient and probably unique location. The case of overlapping bases is handled indirectly by the search routine. The first base coordinates that fall within the range of the pick dictate the component to be deleted. After deletion is complete, the entire new linkage is displayed. The pick area for deletion of interference panels is a rectangle formed by the maximum and minimum coordinates of the panel. Interference panel coordinates are searched before linkage components, and, as with linkage components, the first panel in the list with the correct pick area is deleted.

After deletion of an internal (i.e. not the last) component is complete, the remaining linkage is connected together as if the component never existed. The connection rule is that the "driver side" (i.e. the driver and components on the driver's side of the void resulting from the deletion of an internal component) do not "relocate" (i.e. their coordinates remain constant). The "nondriver" side is relocated to join the rest of the linkage. Naturally, interference panels remain unchanged.

Other possible example linkages are shown in Figures 2-5 through 2-7. Figure 2-7 illustrates a linkage composed of every component available on the menu all driven by a

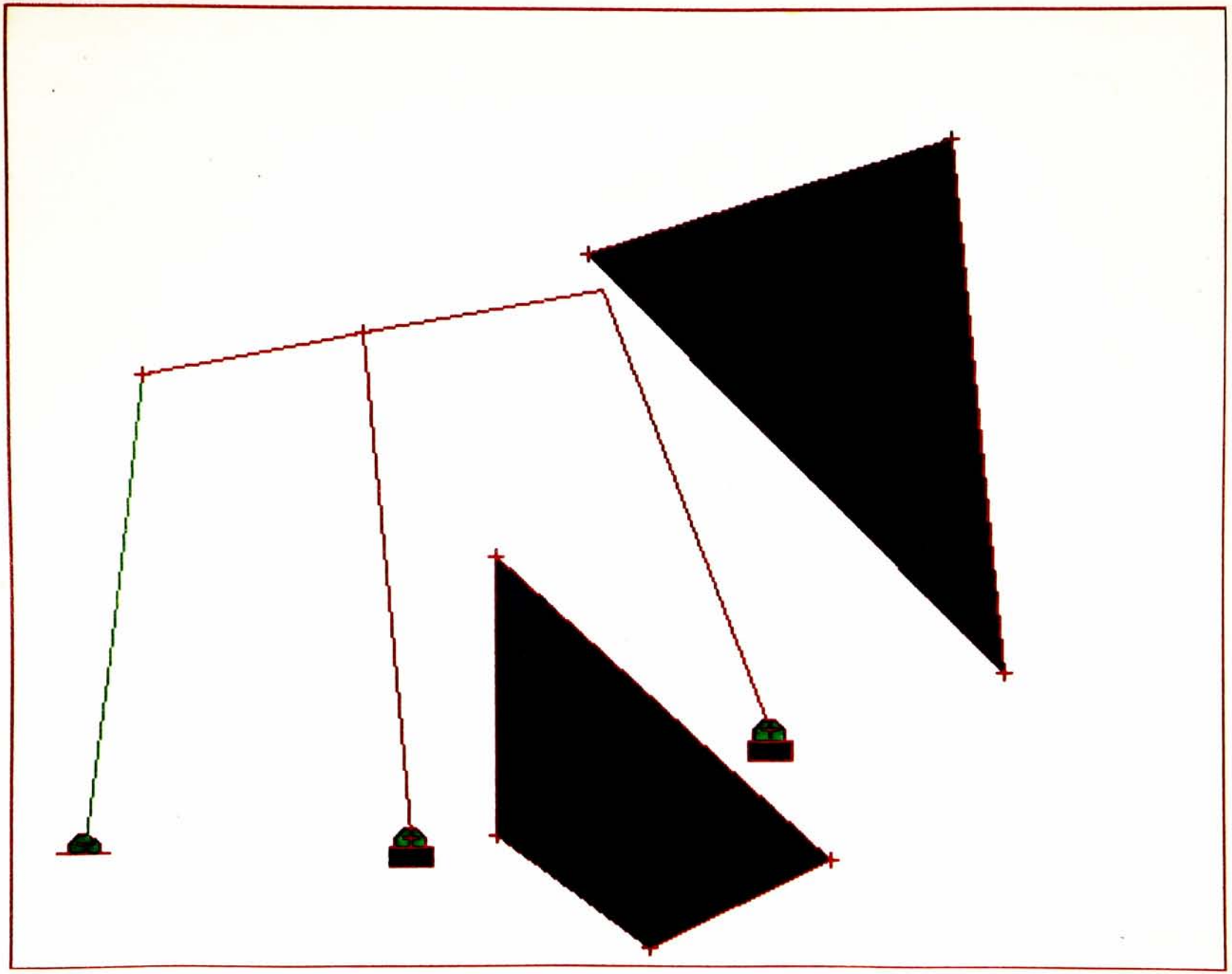


Figure 2-4 LKSP Components, Rotating Driver, and Two Simple Dyads.

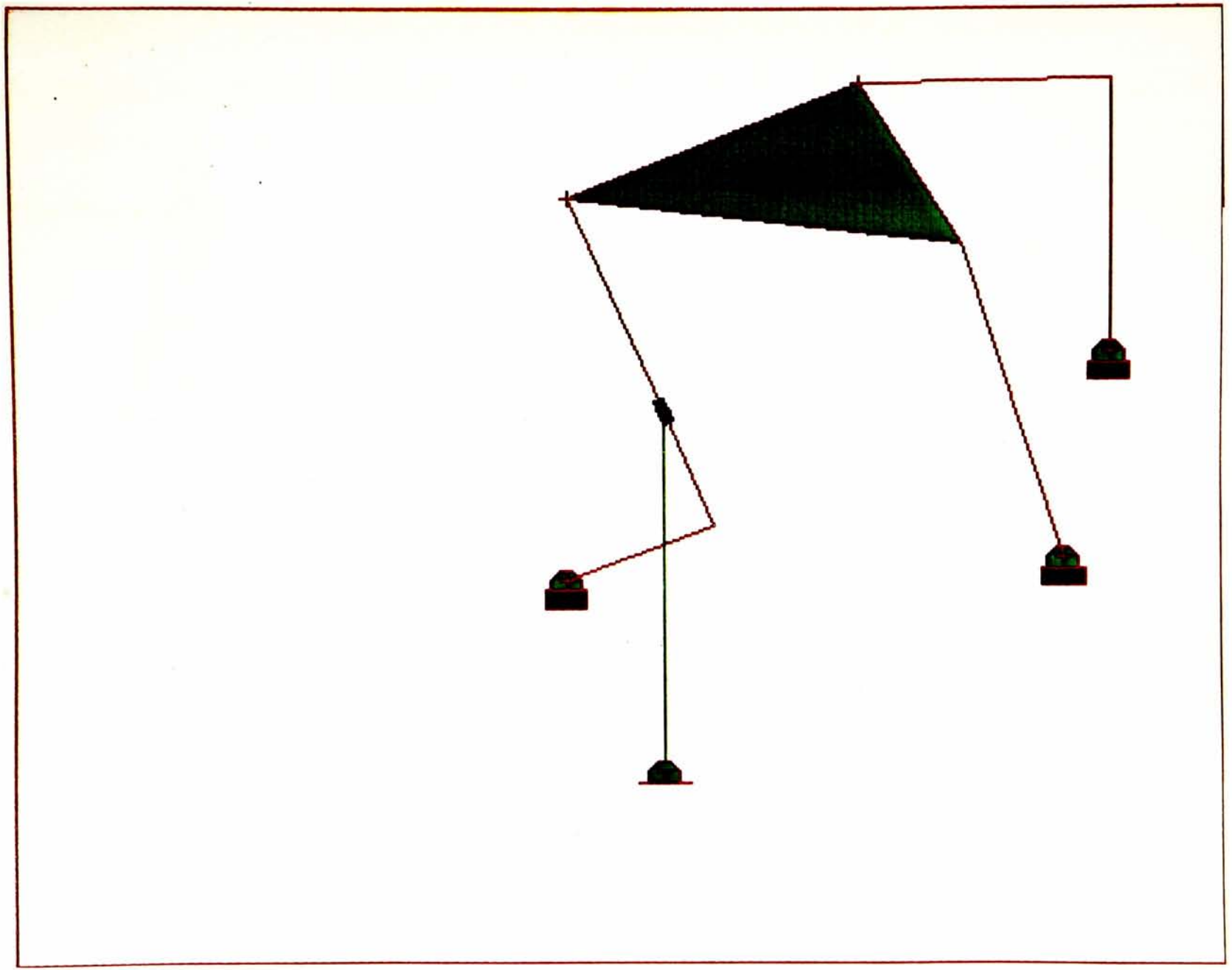


Figure 2-5 LKSP Components, Rotating Driver, Oscillating-Slider, Dyad-2 and Dyad-1.

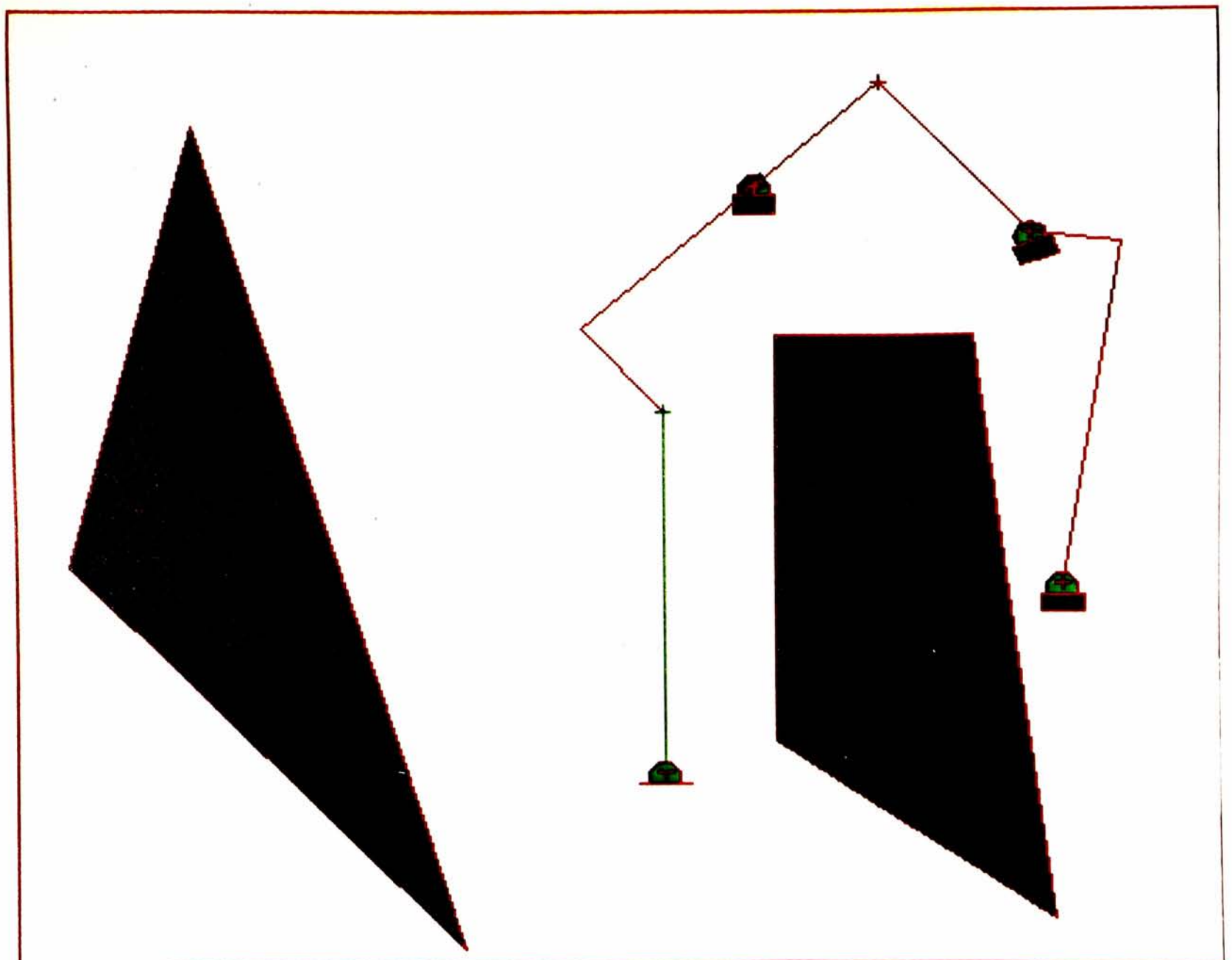


Figure 2-6 LKSP Components, Rotating Driver, Rotating Guide, Slider and Dyad-1.

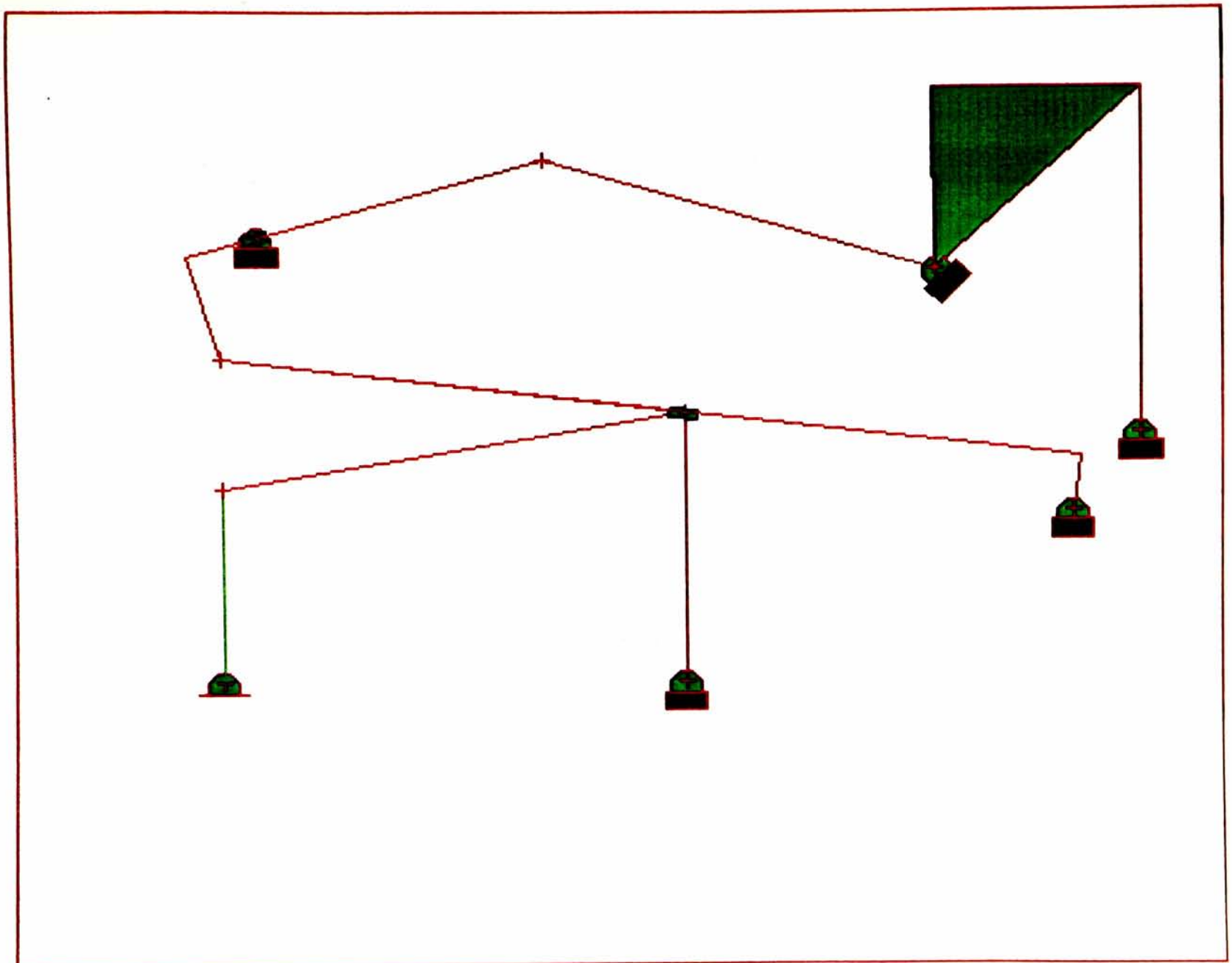


Figure 2-7 Rotating Driver with all LKSP Components.

rotating driver. The input point of a component is shown in the menu icon by a red arrow (Figure 2-3).

Interactive linkage movement calls on the kinematic analysis capability of LKSP. Interactive linkage movement is believed to be a unique feature of LKSP not previously implemented in other software. Figure 2-8 illustrates linkage movement for a simple dyad moved by a rotating driver. The user first picks the "Move" function. Then, a position near the driver is picked. The linkage moves in the direction dictated by the location of the pick relative to the previous position of the driver. In this case the driver starts at position 1 and moves counterclockwise to position 2. The motion of the linkage is not traced (trace function is off). The fine grid function (Grid-f) was "on," but the grid is not visible on the drawing because the white color of the grid is not reproduced by the ink jet plotter. A message tells the user that motion is complete. As the linkage arrives at its new position, it moves through an interference panel. Interference is visible, but the linkage is allowed to pass through.

Another example of linkage motion is illustrated in Figure 2-9. The linkage consists of a rotating driver moving a rotating guide. The trace function was "on." At the start of motion, the linkage is at the leftmost position. The driver moves clockwise toward the picked target position (point added in black). The yellow curved lines are the traces of the motion. Once again, as the linkage arrives at its new position, it moves just into the edge of the right interference panel. The occurrence of interference is obvious.

In Figure 2-10, a dyad-2 is driven by a rotating driver through part of its motion cycle. The linkage starts at position 1, and the driver moves counterclockwise to the target point at position 2. All three corner points on the dyad-2 are traced, allowing complete visualization of the motion. Also the panel representing one leg of the dyad-2 is shown in a different color (red) during motion to allow differentiation between its position in transit and its still position. Notice the path reversal of the external tracer point on the dyad-2. In an actual design case, this point might represent a positioning component, and its motion would probably be of primary interest.

In Figure 2-11, the user attempts to "drive" the linkage past its inherent motion limit. In other words, motion becomes impossible because of the configuration of the linkage; the linkage gets "bound-up" and motion stops. In this case, the driver moves from right to left (counterclockwise), and the dyad-2 follows along, carrying with it the rotating guide at the top. When the rotating guide hits the end of its shaft, binding occurs; motion of the entire linkage ceases, and

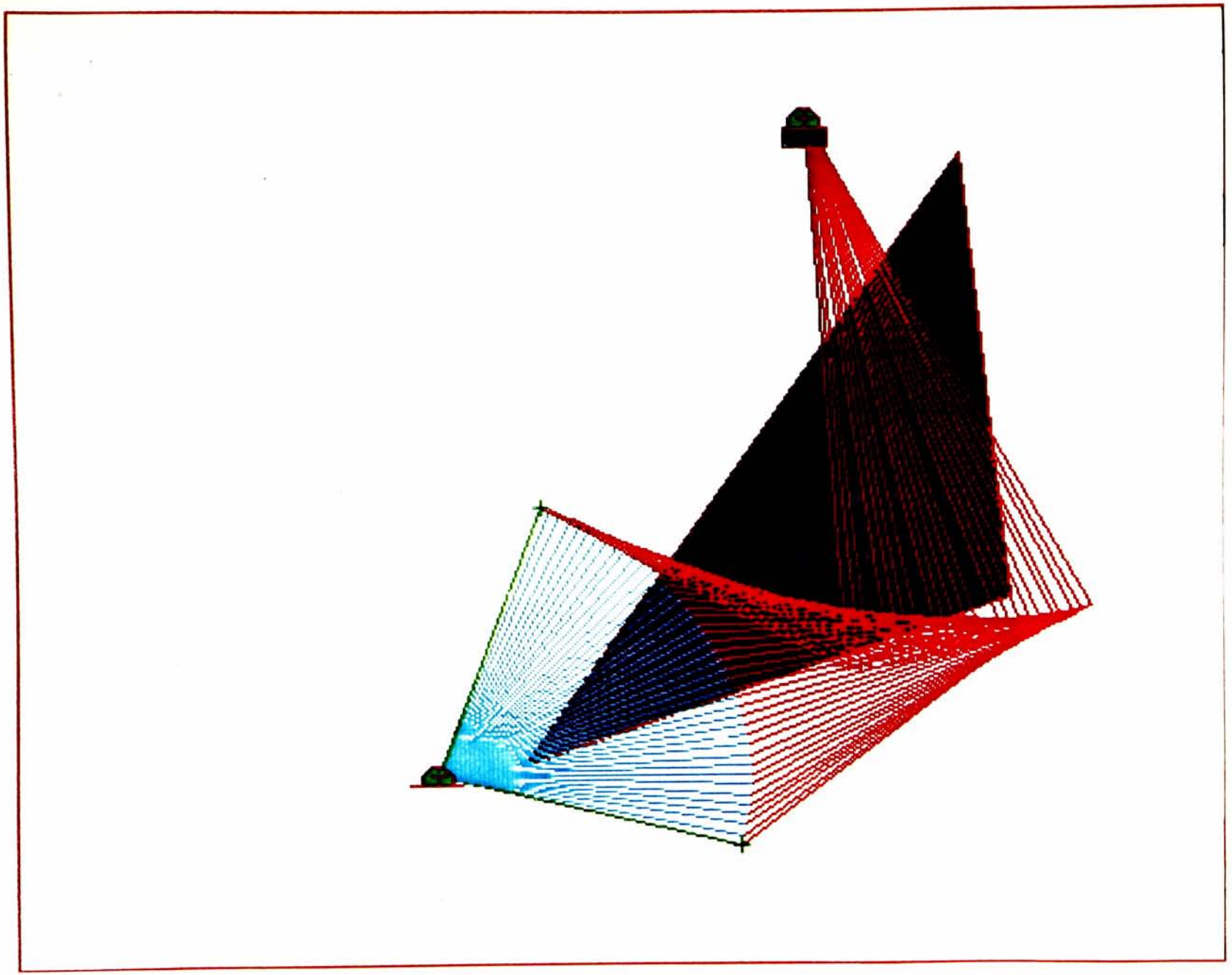


Figure 2-8 Rotating Driver with Dyad-1, in Motion, no Trace.

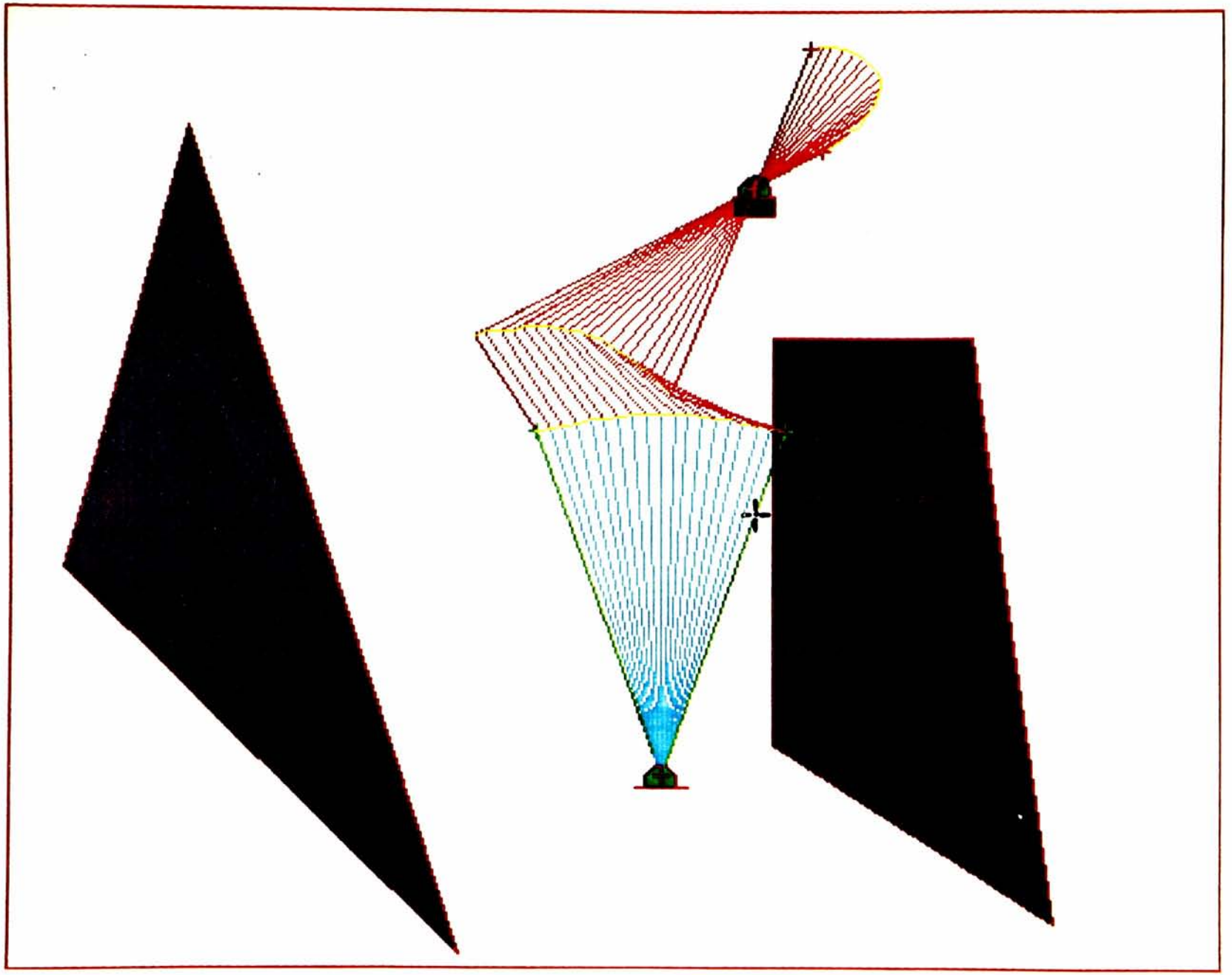


Figure 2-9 Rotating Driver with Rotating Guide, in Motion, Trace "on", 2 Ipanels.

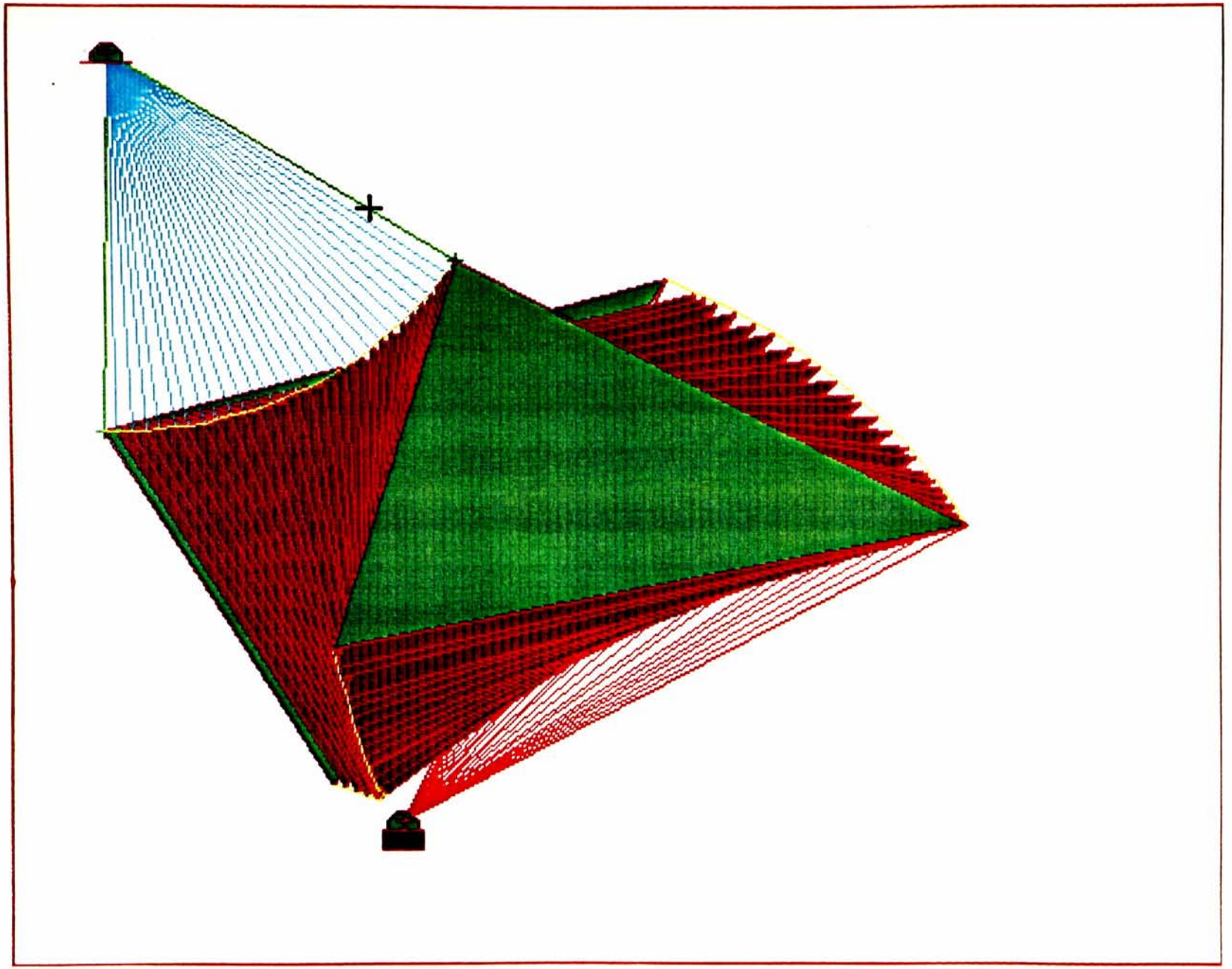


Figure 2-10 Rotating Driver with Dyad-2, in Motion, Trace "on".

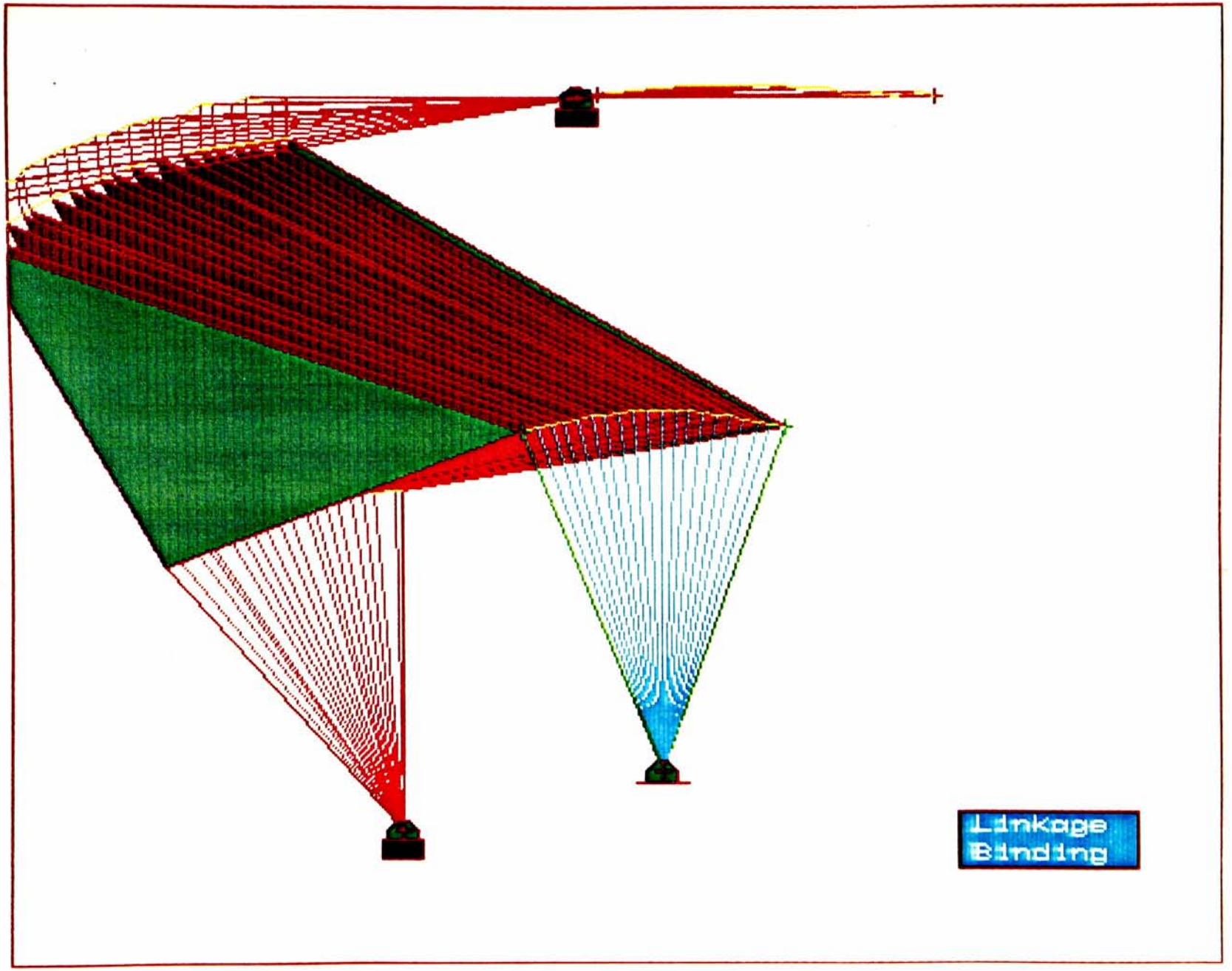


Figure 2-11 Rotating Driver with Dyad-2 and Oscillating Slider, in Motion, Trace "on", Linkage Binding.

a warning panel appears stating that the linkage is binding. The warning message has a blinking background color to attract the attention of the user.

Motion of a translating driver connected to a simple dyad is illustrated in Figure 2-12. As the driver moves from the upper left toward the target point at the lower right, the dyad connection point starts at point 1 (added in black), moves to point 2, then reverses direction and rotates back to point 3 This motion is very obvious when viewed during the animation on the screen of the CRT. The trace path of the translating driver is illustrated as a series of light blue points.

The red lines showing the links of the dyad at each location show the "history" of its position on the screen. Their spacing illustrates the speed of the movement. In this case, the dyad moves more rapidly after the motion reversal. Notice how close the toggle point of the dyad (point 2) comes to the edge of one of the interference panels.

Another example of translating driver motion is illustrated in Figure 2-13. In this case the driver is connected to a rotating guide with a very small offset. Again, the driver moves from the upper left toward the target point at the lower right. The rotating guide moves accordingly, tracing the path shown in the illustration. Shortly after motion begins, the rotating guide impinges on the lower interference panel (as shown by the arrow added in black). The effect of increasing rotational velocity is again obvious from the spacing of the red lines showing the links of the dyad at each location. In this case, a part of the motion traces has been clipped. The motion algorithms (see section 1.3), which are accessed to create animation, were completely verified with analytical calculations under all kinematic conditions including binding and combined motion of multiple component linkages. This completes discussion of user input and output.

2.1.4 System Files

The following is a description of the files created, accessed and maintained by the system. Files are required to support the menu and to keep track of components within the system. These will interface with the other files required to support graphics, i.e. the lists of segments and the DPU (Display Processing Unit) code list as shown in Figure 2-14. There is also a file dedicated to handling the records for the linkage while it is in motion. Graphics is discussed in section 2.2.1.

The Menu Items List supports the menu. Its function is to correlate the menu and its structure to the segments list. The structure of these files is shown in Figure 2-15. Also shown

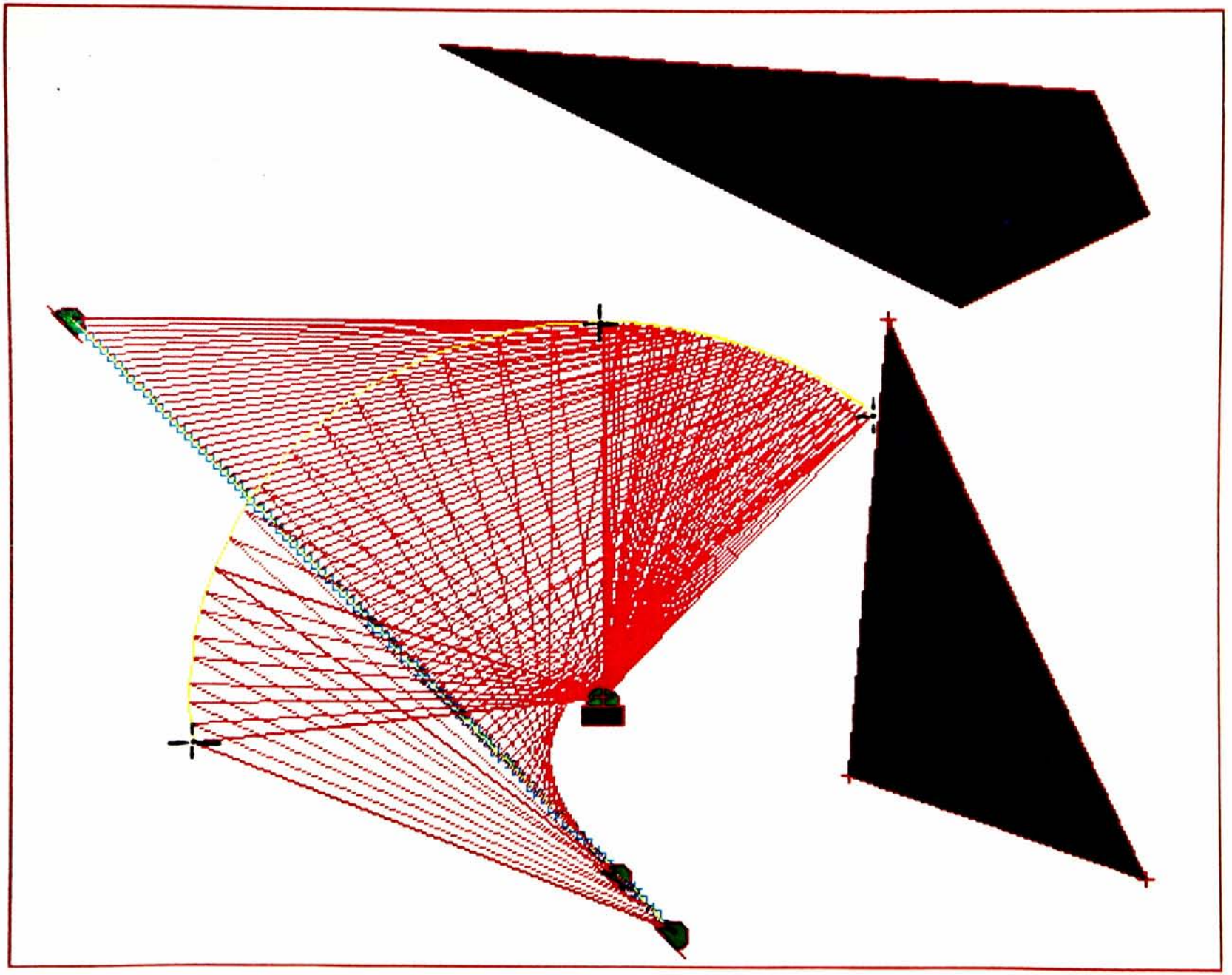


Figure 2-12 Translating Driver with Dyad-1, in Motion, Trace "on", 2 Ipanels.

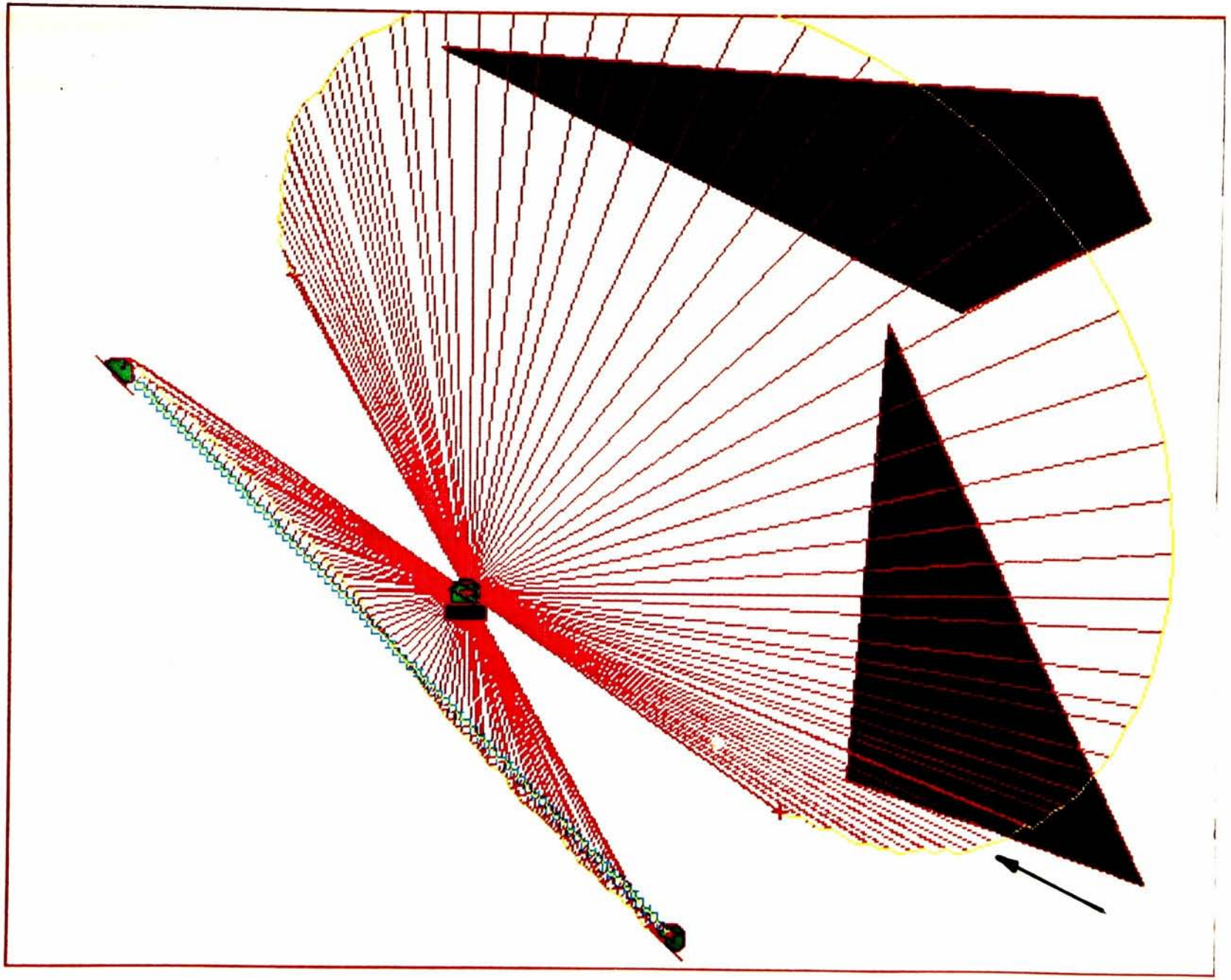


Figure 2-13 Translating Driver with Rotating Guide in Motion, Trace "on", 2 Ipanels.

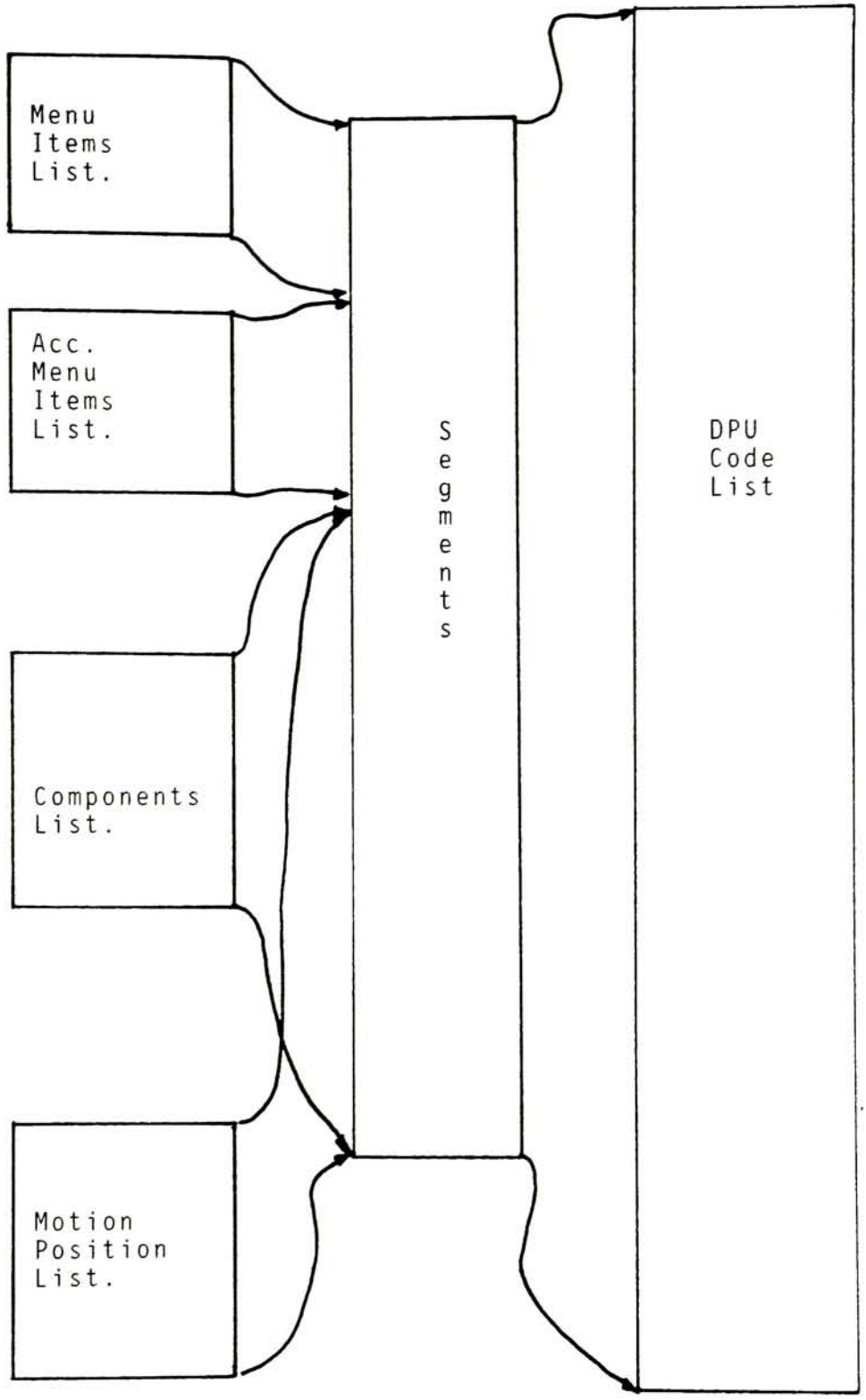


Figure 2-14 System Files Hierarchy.

Menu Items List.

#	name	segno	pick boundary xmin	ymin	xmax	ymax
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Acc. Menu Items List.

#	name	segno
.	.	.
.	.	.
.	.	.

Components List.

#	name	segno base	segno rest	out pt.	position coord-4	lengths & angle	Moa 1&2	pick boundary coord-2
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Motion Position List.

#	name	segno rest	out pt.	current position coord-4	last position coord-4	lengths & angle	Moa 1&2	binding y/n
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

I-Panels List.

#	name	segno	position coord-4	pick boundary coord-2
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Figure 2-15 System Files Structure.

in Figure 2-15 is the Components List. Its function is to correlate the linkage components to the segments list. It also contains other information needed for each component such as its type, location, trace points and an angle (associated with either the component's collar or its included angle for dyad-2).

The structure of the Motion Position List is also shown in Figure 2-15. The Motion Position List is an expanded version of the Components List which accounts for the component's previous location (for tracing), and other factors only required during movement of the linkage. It also correlate the linkage components to the segments list, but only during motion animation.

This completes the discussion of system files as well as the functional description of LKSP.

2.2 System Description

The following is a top-level, description of the system's architecture and flow. This description consists of system organization and module hierarchy, data flow, equipment configuration and implementation tools.

2.2.1 System Organization

A set of 2-D graphics primitives per the 1979 A.C.M. CORE Standard (53), which were created for R.I.T. course ICSS-770, Computer Graphics, were used. These primitives support segmentation and assure device independence (54, 55). A new device driver was created to support the hardware which was used for this project. Hardware is discussed in section 2.2.4.

At the highest level, system organization consists of modules which support the menu and those which support each of the earlier outlined functions. These procedures are accessed by a main software module (LKSPmain) as illustrated in Table 2-1. Source listings of some software modules are included in Appendices II through IV.

Prep is a "preparation" procedure which takes care of all initial requirements of the system. It is the first procedure called by main. Procedures accessed by Prep are outlined and further subdivided in Tables 2-2 through 2-5.

Any user request is handled by "do" routines. In essence, they "do" everything required to implement a user request including

Table 2-1 Procedures accessed by main software module, LKSPmain.

LKSPmain		Main operating program.
	Prep	Preparation procedure, initial requirements.
	DisplayGinEnable	Send "ginenable" to terminal.
	GetMenuItem	Report on menu item picked.
	doerrorpick	Detect menu pick errors.
	doGridf	Fine grid.
	doGridc	Coarse grid.
	doRotatingDriver	Rotating Driver.
	doTransDriver	Translating Driver.
	doReplaceDriver	Replace Driver.
	doDyad1	Dyad-1.
	doDyad2	Dyad-2.
	doOscSlider	Oscillating Slider.
	doRotatingGuide	Rotating Guide.
	doSlider	Slider.
	doIpanel3	Interference
	doIpanel4	panels.
	doDelete	Deletion.
	doTrace	Tracing.
	doMove	Movement.
	doClean	Clean up display.
	doEndSession	End program.

Table 2-2 Procedures accessed by Prep.

The following procedures are intended for access by the Prep procedure only. Prep is a preparation procedure which handles initial requirements and is the first procedure called by main.

The procedures consist of segmentation and graphics primitives which describe parts of the menu and perform initialization tasks.

Procedures have a beginning keyword which is intended to indicate its primary function as follows

make	creation of a segment with graphics primitives, visibility is turned off.
Display	visibility of an already created segment is turned on and the screen is updated.
initi	initialization of a data structure such as the components list.

The procedures accessed by Prep are

makeInitGraphics
DisplayInitGraphics

makeBorder1
DisplayBorder1

makeBorder2
DisplayBorder2

initiComponentsList

makeMenu
DisplayMenu

makeAccMenu

makeGreeting
DisplayGreeting

makeGinEnable

makeFinishUp

Table 2-3 Procedures accessed by makeMenu software module.

The following procedures are intended for access by the makeMenu procedure only. They consist of segmentation and graphics primitives which describe parts of the menu. Manipulation of the CTM is required to "place" the menu item wherever required.

During this creation phase, visibility is turned off. "Display" procedures are used to turn on visibility and update the display.

There is also a procedure to create the menu items list.

The procedures are

CreateMenuItemsList
initiMenuItemsList

makeMenuBorder

makeMenuGridf
makeMenuGridc

makeMenuRotatingDriver
makeMenuTransDriver
makeMenuReplaceDriver

makeMenuDyad1
makeMenuDyad2
makeMenuOscSlider
makeMenuRotatingGuide
makeMenuSlider

makeMenuIpanel3
makeMenuIpanel4

makeMenuDelete

makeMenuTrace

makeMenuMove

makeMenuCleanup

makeMenuEndSession

Table 2-4 Procedures accessed by DisplayMenu software module.

The following procedures are intended for access by the DisplayMenu procedure only. These "Display" procedures are used to turn on visibility of previously created graphics segments (makeMenu) and update the display.

DisplayMenuBorder

DisplayMenuGridf
DisplayMenuGridc

DisplayMenuRotatingDriver
DisplayMenuTransDriver
DisplayMenuReplaceDriver

DisplayMenuDyad1
DisplayMenuDyad2
DisplayMenuOscSlider
DisplayMenuRotatingGuide
DisplayMenuSlider

DisplayMenuIpanel3
DisplayMenuIpanel4

DisplayMenuDelete

DisplayMenuTrace

DisplayMenuMove

DisplayMenuCleanup

DisplayMenuEndSession

Table 2-5 Procedures accessed by makeAccMenu software module.

The following procedures are intended for access by a second version of the makeMenu procedure called "makeAccMenu." This procedure is intended to create the same graphics primitives as in makeMenu, but with different color (and segmentation).

As with makeMenu, during the creation phase, visibility is turned off and Display procedures are used to turn on visibility and update the display. The resulting "accented color" version of menu items is intended to be used by "do" procedures to acknowledge a user pick.

The procedures are

makeAccMenuGridf
makeAccMenuGridc

makeAccMenuRotatingDriver
makeAccMenuTransDriver
makeAccMenuReplaceDriver

makeAccMenuDyad1
makeAccMenuDyad2
makeAccMenuOscSlider
makeAccMenuRotatingGuide
makeAccMenuSlider

makeAccMenuIpanel3
makeAccMenuIpanel4

makeAccMenuDelete

makeAccMenuTrace

makeAccMenuMove

makeAccMenuCleanup

1. Request acknowledgment.
2. Acquisition of the required user input.
3. Manipulation of input.
4. Creation and processing of output.
5. File updates.
6. User interaction via prompts and error messages.
7. Error recovery.

There is one "do" procedure for every menu item and one additional procedure for handling ambiguous menu picks (such as a menu pick not on the menu). The "do" procedures are outlined and further subdivided in Tables 2-8 through 2-13.

Miscellaneous software modules consisting of graphics primitives, "icons," message handling routines and graphics I/O at the lowest level are described Tables 2-6 and 2-7. A system organizational chart is presented in Figure 2-16. This chart pictorially describes software module hierarchy, intermodule calling sequences and communications for modules created as part of LKSP. A list of graphics primitives appears in Appendix IV. In addition, a summary of code by function and file size, for source and executable modules, appears in Table 2-14.

2.2.2 System Data Flow

A pictorial description of information flow among the various top-level software modules is presented as a system data flow chart in Figure 2-17. Input is strictly handled by the main operating program. Output is handled at various levels.

2.2.3 Equipment Configuration

The hardware to be used for this project consists of a Tektronix 4109 color graphics display terminal connected to a VAX-Cluster, consisting of two VAX 11/785 superminicomputers running under the VMS operating system (56). The equipment configuration is shown in Figure 2-18.

The VAX-Cluster system is located in a central scientific computing department devoted primarily to running large analysis packages for engineering applications. The graphics terminal is located in another building and is connected to the VAX system via a dedicated RS-232 line with a modem capable of 9600 baud transmission speed.

The Tektronix 4109 color terminal (57 to 60) incorporates a 60 Hz, noninterlaced, raster-scan display. The screen is about 14 inches horizontally by 10.5 inches vertically (i.e. a 19-inch diagonal screen)

Table 2-6 Miscellaneous software modules.

The procedures below consist of graphics primitives only. They are filler panels of background color. These are intended to be accessed by procedures which create parts of the menu.

Manipulation of the CTM is required to "place" the panels wherever required. Also, creation of segments is assumed.

MenuPanel1
MenuPanel2
MenuPanel3
MenuPanel4
MenuPanel5
MenuPanel6
MenuPanel7

The following "icons" consist of graphics primitives only. These can be accessed by just about any procedure which displays components or the entire linkage.

Manipulation of the CTM is required to "place" the icon wherever required. Also, creation of segments is assumed.

DriverBase
ComponentBase
Text
Collar(angle)
CrosshairRel

Table 2-7 Miscellaneous software modules (continued).

These procedures are required to dump messages (prompts or error messages) of various lengths. They can be accessed by just about any procedure which requires them.

Prompt20
Prompt30
Prompt40
Prompt50
Bell
Error20
Error30
Error40
Error50

Function of DisplayGinEnable and getMenuItem.

Access to all features ("do" routines) is through menu input (selection) which is handled by the main operating module of LKSP. Menu input is supported by two procedures, DisplayGinEnable and getMenuItem.

DisplayGinEnable displays the cursor and prepares for graphics input. Procedure getMenuItem receives the "gin point" and interprets it as a menu item or an error.

Creation of the graphics primitives contained in "Display" procedures via "make" procedures was previously explained.

Table 2-8 Function of the "do" software modules.

The "do" procedures are accessed from the main by user selection through the menu only. The "do"procedures are the functional and control workhorses of the LKSP package. In essence, they "do" everything required to implement a user request including

1) Request acknowledgement
2) Aquisition of the required user input
3) Manipulation of input
4) Creation and processing of output
5) File updates
6) User interaction via prompts and error messages
7) Error recovery

There is one "do" procedure for every menu item and one additional procedure for handling ambiguous menu picks (such as a menu pick not on the menu). The "do" procedures include

doerrorpick	Detect menu pick errors.
doGridf	Fine grid.
doGridc	Coarse grid.
doRotatingDriver	Rotating Driver.
doTransDriver	Translating Driver.
doReplaceDriver	Replace Driver.
doDyad1	Dyad-1.
doDyad2	Dyad-2.
doOscSlider	Oscillating Slider.
doRotatingGuide	Rotating Guide.
doSlider	Slider.
doIpanel3	Interference panels.
doIpanel4	
doDelete	Deletion.
doTrace	Tracing.
doMove	Movement.
doClean	Cleanup display.
doEndSession	End program.

Table 2-9 Procedures Accessed by "do" routines, DisplayAccMenu modules.

The following procedures support the "do" procedures for request acknowledgement, i.e. telling the user that a menu pick is good and the request for action will be honored.

These procedures display selected menu items in an accented color, thereby acknowledging the request. As "Display" procedures, they turn on visibility of previously created graphics segments and update the display.

As previously mentioned, the graphics primitives are created by a procedure called "makeAccMenu" and during this creation phase, visibility is turned off. The procedures include

DisplayAccMenuGridf
DisplayAccMenuGridc

DisplayAccMenuRotatingDriver
DisplayAccMenuTransDriver
DisplayAccMenuReplaceDriver

DisplayAccMenuDyad1
DisplayAccMenuDyad2
DisplayAccMenuOscSlider
DisplayAccMenuRotatingGuide
DisplayAccMenuSlider

DisplayAccMenuIpanel3
DisplayAccMenuIpanel4

DisplayAccMenuDelete

DisplayAccMenuTrace

DisplayAccMenuMove

DisplayAccMenuClean

DisplayAccMenuEndSession

Table 2-10 Procedures Accessed by "do" Routines, "get" Modules.

The following procedures support the "do" procedures for acquisition of user input, i.e. interactively guiding the user through the input (location picks) of information. Those procedures requiring coordinate input data will use the DisplayGinEnable and getGinPoint procedures.

Interaction includes telling the user that coordinate picks are acceptable (within the graphics display area) and display of linkage components or parts thereof. Previously created graphics segments (icons) are used in the display of components.

Procedure getUwindowdata is the only case where user input is via keyboard. The following "do" procedures require user input

getRotatingDriverdata
getTransDriverdata
getReplaceDriverdata

getDyad1data
getDyad2data
getOscSliderdata
getRotatingGuidedata
getSliderdata
getIpanel3data
getIpanel4data

getDeletedata

getMovedata

Table 2-11 Procedures accessed by "do" Routines, support procedures, Part 1.

One procedure, doerrorpick, accesses only graphics primitives, prompts and error message handling routines Most "do" procedures access a number of support procedures which handle the manipulation of user input and files specific to their particular function.

These routines create and display a user requested grid:

```
doGridf
        makeGridf
        DisplayGridf
doGridc
        makeGridc
        DisplayGridc
```

These routines compute reasonable driver motion increments used for animation of the linkage. The range of acceptable values for motion increments was determined as part of this work.

```
doRotatingDriver
        FindIncRotatingDriver
doTransDriver
        FindIncTransDriver
```

Table 2-12 Procedures accessed by "do" Routines, support procedures, Part 2.

The procedures updateComponentsList and updateComponentsDisplay are used during the user creation of components to create and display component parts of the linkage.

doDyad1	
doDyad2	
doOscSlider	updateComponentsList
doRotatingGuide	updateComponentsDisplay
doSlider	

The procedures updateIpanelsList and DisplayIpanels are used during the user creation of interference panels to create and display the Ipanels.

doIpanel3
- updateIpanelsList
- DisplayIpanels

doIpanel4
- updateIpanelsList
- DisplayIpanels

The doDelete procedure is supported by a number of routines to scan the ComponentsList and find the user picked component, delete the user picked component from the ComponentsList, and DPU code list and finally to again display the entire linkage.

doDelete
- GetLinkageComponent
- GetIpanel
- DeleteLinkageComponent
- DeleteIpanel
- DisplayLinkage

Table 2-13 Procedures accessed by "do" Routines, support procedures, Part 3.

The doMove procedure is the key motion visualization routine in LKSP. It is supported by a number of procedures to compute new positions for each component type.

The updateMotionPositionList and DisplayMovingLinkage procedures iteratively display the linkage animation.

doMove

- CalcNewPosRotatingDriver
- CalcNewPosTransDriver
- CalcNewComponentPositions
  - CalcNewPosReplaceDriver
  - CalcNewPosDyad1
  - CalcNewPosDyad2
  - CalcNewPosOscSlider
  - CalcNewPosRotatingGuide
  - CalcNewPosSlider

The last "do" routines are:

- doClean
  - DisplayLinkage
- doEndSession
  - DisplayFinishUp

Table 2-14 LKSP Code Summary.

The following summary lists the approximate number of lines of Pascal source code, file sizes (VMS blocks) for source and executable modules, and the approximate percent of the total project time (for code creation and debug) spent on each main functional capability within LKSP.

Function.	Source code, lines.	File sizes, blocks source.	obj.	Percent of time for code creation.
graphics primitives and device driver.	2268	167	61	25
menu creation and support.	2989	132	83	35
"do" procedures linkage motion, creation and file maintenance.	4521	244	84	30
Main, including declarations and Prep.	1980	103	28	10

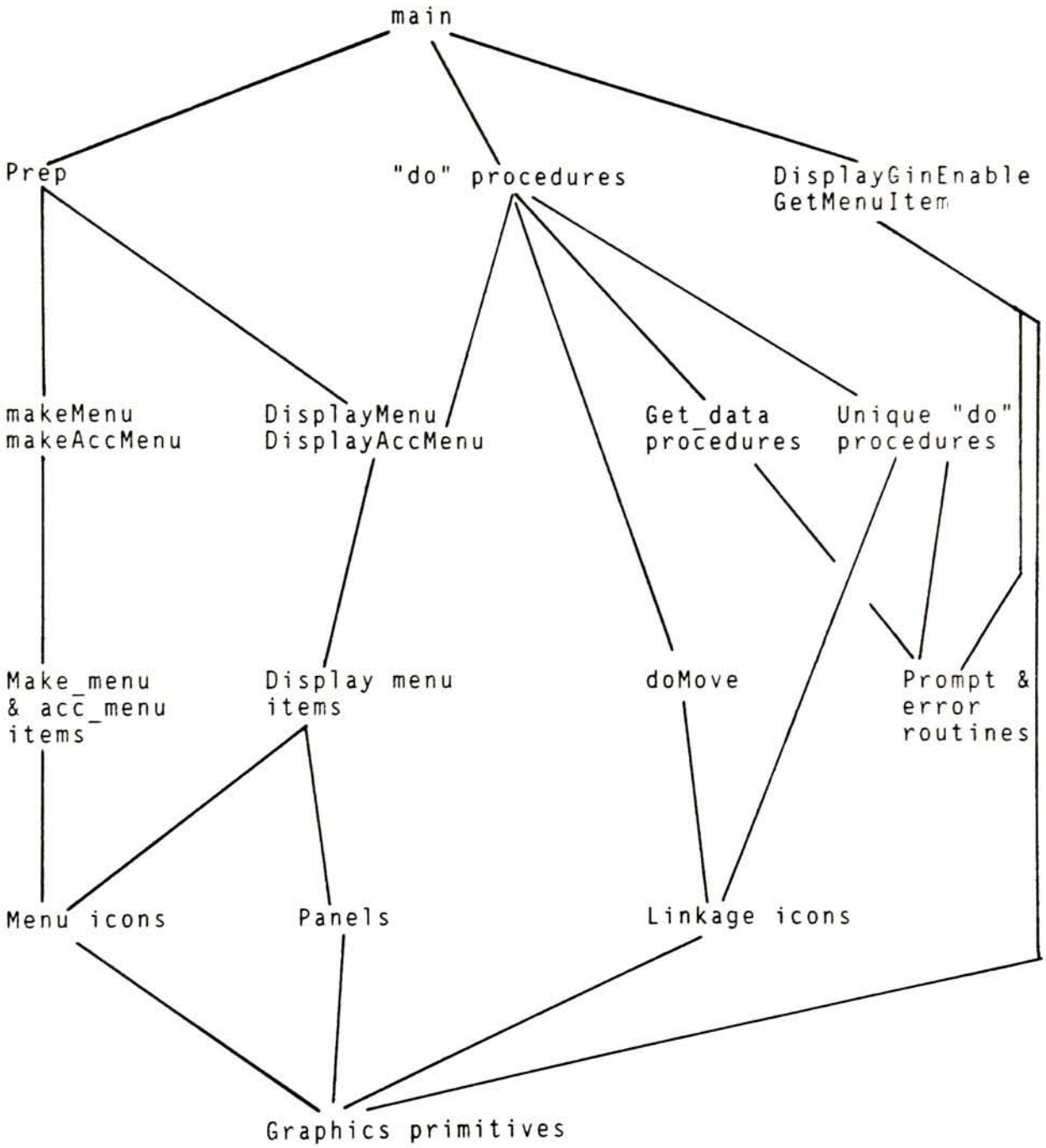


Figure 2-16 System Module Hierarchy.

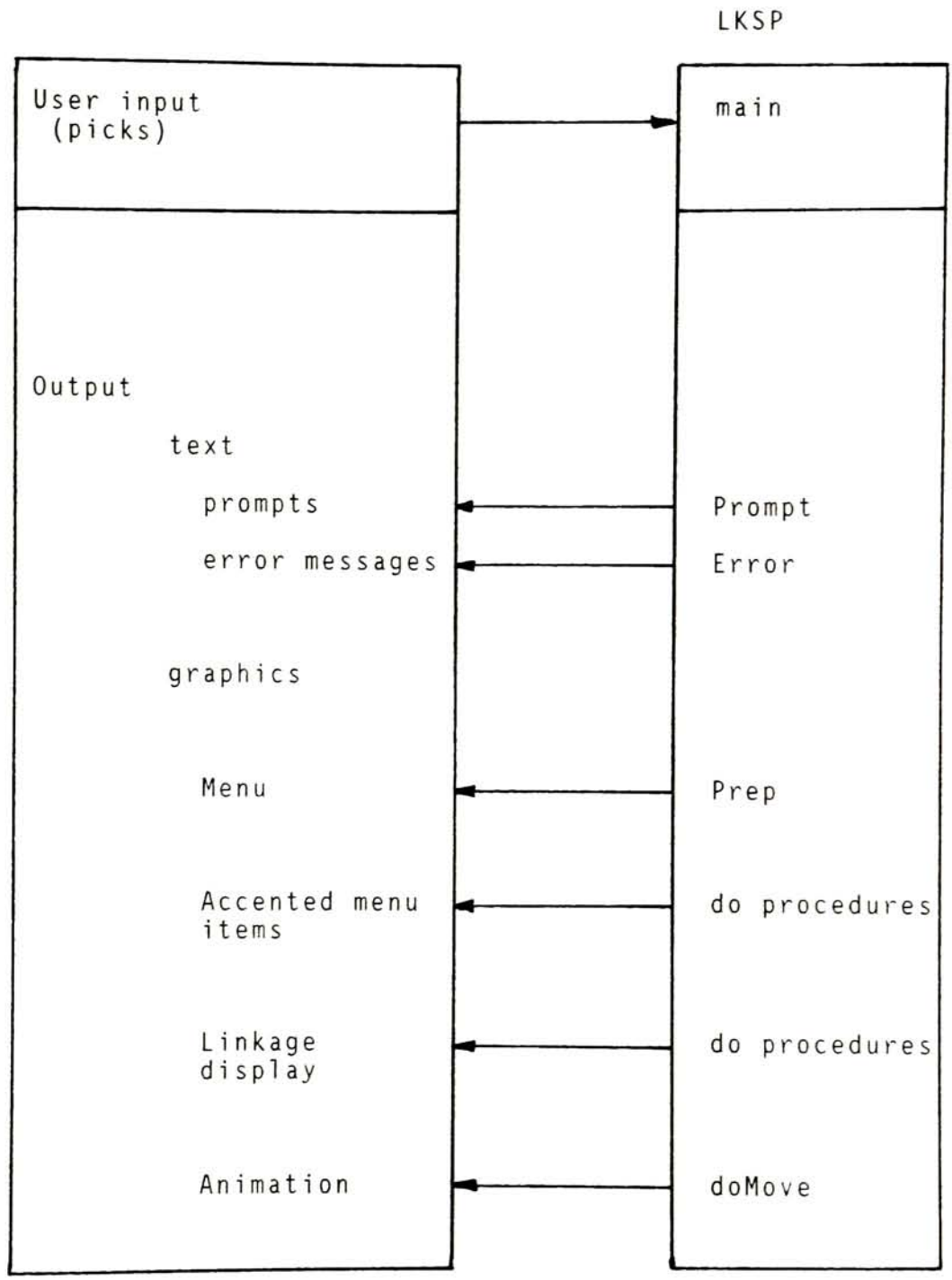


Figure 2-17 System Data Flow, Top Level.

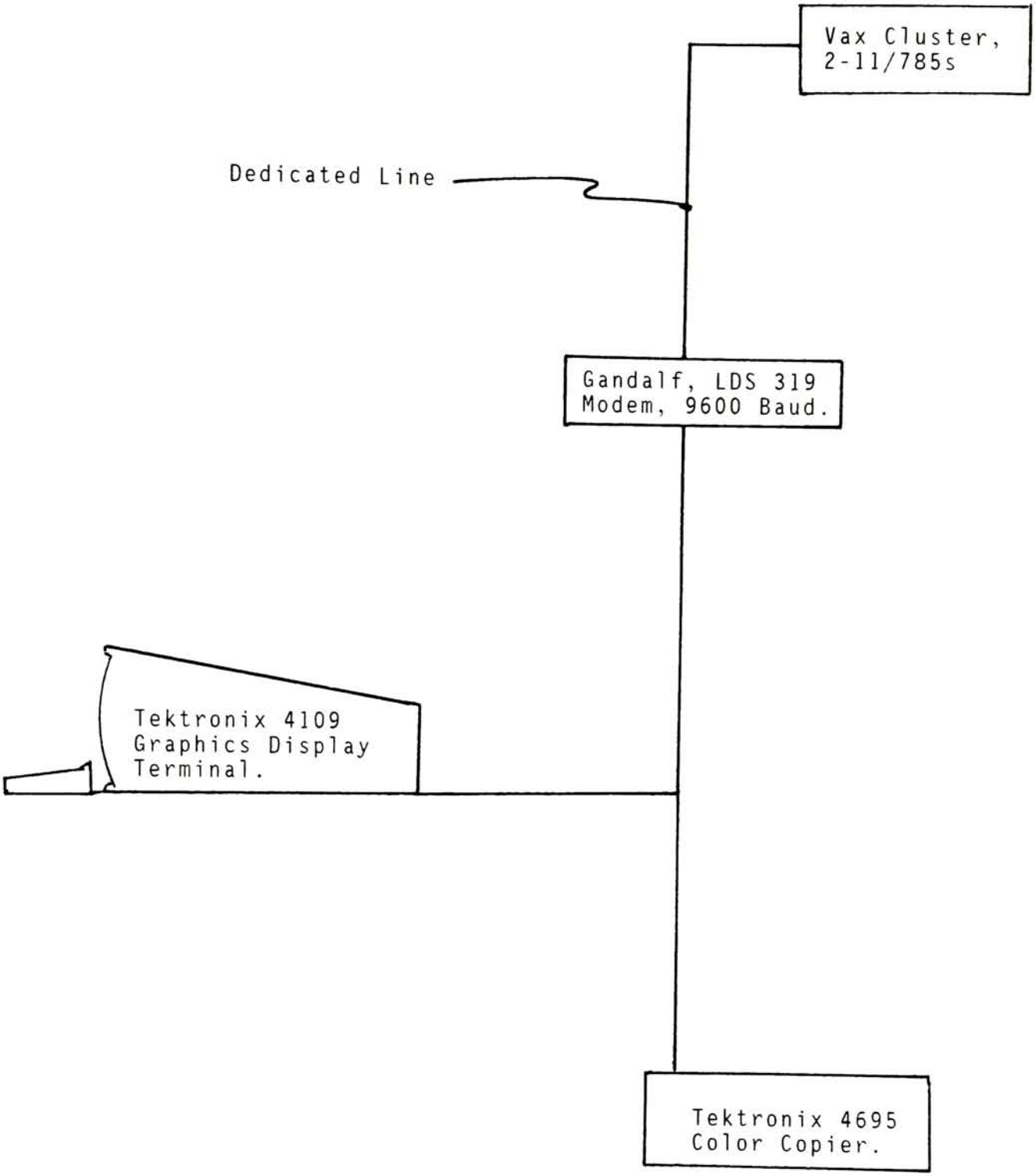


Figure 2-18 Equipment Configuration.

The display resolution is 640 (horizontal) by 480 (vertical) pixels. Alphanumerics can be displayed in either the graphics area (i.e. over graphics or with background) or in a host-definable dialog area (from 2 to 32 lines).

There are 4096 distinct color mixtures available and up to 16 can be displayed at one time in the graphics area. Blinking versions of colors are available as well as a variety of line styles and widths. Other features include

1. Local filling of panels with a variety of patterns or color shading.

2. Local segment operations, windows, viewports and zoom and pan.

3. Multiple graphics surfaces.

4. Pixel operations.

All of these features are supported within the Tektronix terminal by a microprocessor (Intel 80186) based system with one megabyte of volatile and nonvolatile local memory. Up to a few years ago, this terminal was considered to be "state of the art."

2.2.4 Implementation Tools

The Pascal language, as defined by Kathleen Jensen and Niklaus Wirth (61) was chosen because

1. It is a structured language making code preparation, de-bug and maintenance relatively easy.

2. A set of device independent graphics primitives have been written by this author in Pascal and are available for use.

3. The target implementation hardware has Pascal available (62, 63) with extensions and other system supported features.

This ends discussion of system description as well as the project description.

3. Evaluation of LKSP, Conclusions and Recommendations.

3. Evaluation of LKSP, Conclusions and Recommendations.

After software creation and debug, LKSP was evaluated by this writer and three other engineers with varying levels of experience in linkage design from little experience to expert. The primary goal of my evaluation was to determine the adequacy of

1. The system design.
2. Correctness of the implementation.
3. The operating efficiency.
4. The hardware used.
5. The theoretical investigation.

Much of my investigation took place during system design (items 1 and 5 above), code creation, test and debug (item 2 above) phases of this project. Summaries of these aspects appear in sections 1 and 2 of this report. In brief, the system design appears to be adequate in all respects. Also, the implementation has been thoroughly tested, and it is correct. Linkage motions are as required (see section 2), and the code is performing all required functions as planned. In addition, the theoretical investigation is believed to be complete and current (see section 1). Items 3 and 4 were primarily evaluated after the code was up and running. These will be discussed in sections 3.3 and 3.4.

The primary goal of the user evaluation was to determine the adequacy of

6. User/computer interface, i.e. ease of use.
7. Usefulness of unique features.

Both evaluations overlapped a great deal with regard to the above mentioned criteria, and many of the user results dealt with items 1 through 5 as well as 6 and 7. Evaluation results can be categorized into the following areas

1. Favorable aspects.
2. Shortcomings.
3. Alternate approaches.
4. Suggestions for future extensions.

It was generally agreed that LKSP offers a unique approach to planar linkage design with the most desirable features being the interactive design of the user/computer interface, the ability to create linkages with great ease, and the ability to observe linkage motion (and potential for interference) interactively. A video tape showing actual linkage creation and animation was created to illustrate the capabilities of the program.

The most commonly cited shortcoming was the limited set of linkage components which LKSP can handle. There were also some aspects of the motion animation which were improved as a result of the user evaluations. In the following sections, a summary (consensus of my evaluation and user feedback) of each remaining aspect of the evaluation is discussed (3.1 3.4) followed by a summary of suggestions for future extensions to LKSP (3.5).

3.1 User/computer Interface

As previously mentioned, the user/computer interface was one of the most favorably received aspects of this prototype package. Linkage creation, deletion, use of grids and handling of drivers were all found to be sensible and well designed. The interface was found to be pleasing and functional. These "human factors" design aspects were well received due to the use of immediate feedback with prompts and error messages, and the consistent handling of interaction. Also, the number of geometric shapes were kept to a minimum, the number of colors was kept low, and line properties were kept constant.

The screen layout was found to be efficient, and added to the overall acceptability of the system, especially with the graphics display area having an aspect ratio of about one. The menu layout, in particular, was determined to be extremely useful with the various functions being appropriately arranged and their meaning being obvious so that no memorization was required. Also, the appearance of the menu was found to be aesthetically pleasing to most, and the icons were clear and easily understood.

3.2 Usefulness of Unique Features

Overall, it was agreed that the ability to create linkages with great ease, and the ability to observe linkage motion interactively were desirable features, and that the implementation of these features in LKSP was reasonable. It was also agreed that the interference panels were useful.

The animation graphics for interactive motion visualization were well received. Some experimentation was required to determine reasonable values for the rotational and translational motion step sizes. The original design included an algorithm which considered the number of components, and in the case of the rotational driver, the length of the driver. This was not a feasible approach. It was found that a constant translational and rotational step size made motion animation more consistent to the user (i.e. it looked the same every time), and was therefore more acceptable. The translational driver step size was

selected as 0.10 inches (actual size). Figures 2-12 and 2-13 were created with a translational step size of 0.10 inch.

The rotational step size required more experimentation. As LKSP currently exists, all angles are resolved to the nearest degree. It was convenient to resolve angles to 1 degree from a computational standpoint. Figure 3-1 shows a rotating driver moving a rotating guide near a number of interference panels. The step size was set at the 1 degree "minimum" with tracing "on." It is desirable to use a small step size for continuity of the trace paths, especially for complex motions. However, Figure 3-1 becomes extremely cluttered, and the motion animation becomes difficult to follow (an alternative approach is discussed later.). A default value of 2 degrees was found to be acceptable for the rotational step size under most circumstances. Figures 2-8 through 2-11 were created with a rotational step size of 2 degrees.

During motion animation of type-2 dyads with the triangular shaped leg, it was determined that the display can become cluttered, and the trace paths can be obscured because the filled triangular panel often requires much of the available space on the screen (see Figures 2-10 and 2-11). The animation display algorithms were modified to show these triangular panels unfilled during motion. An example of this approach is shown in Figure 3-2. The outline of the triangular panels was added by hand (in black) after the creation of the ink-jet plots, because the available plotter translates the color used for the outline (light gray) to white. It is obvious, however, that these outlines were drawn due to the white lines appearing in the original panel at the start position. This method of motion representation for type-2 dyads was found to be much better than the original method of recreating the filled panel at each motion step.

As a direct result of the above experimentation, another method of portraying linkage motion was devised in which only the trace is plotted at each position. An example of this technique for a simple crank-slider is shown in Figure 3-3. The step size of the rotating driver was reduced to the "minimum" value of 1 degree to achieve more continuous path display. During animation on the screen, the user sees only the yellow trace lines as they are drawn. At the motion target point, the complete linkage is redrawn in its new position. The motion animation is faster with this method than it is when the linkage is redisplayed. It also shows the tracer paths more clearly. However, it is possible to lose track of linkage motion with this technique. Figure 3-4 is offered as an example in which the actual behavior of the dyad-2 is somewhat lost, although the path of the tracer

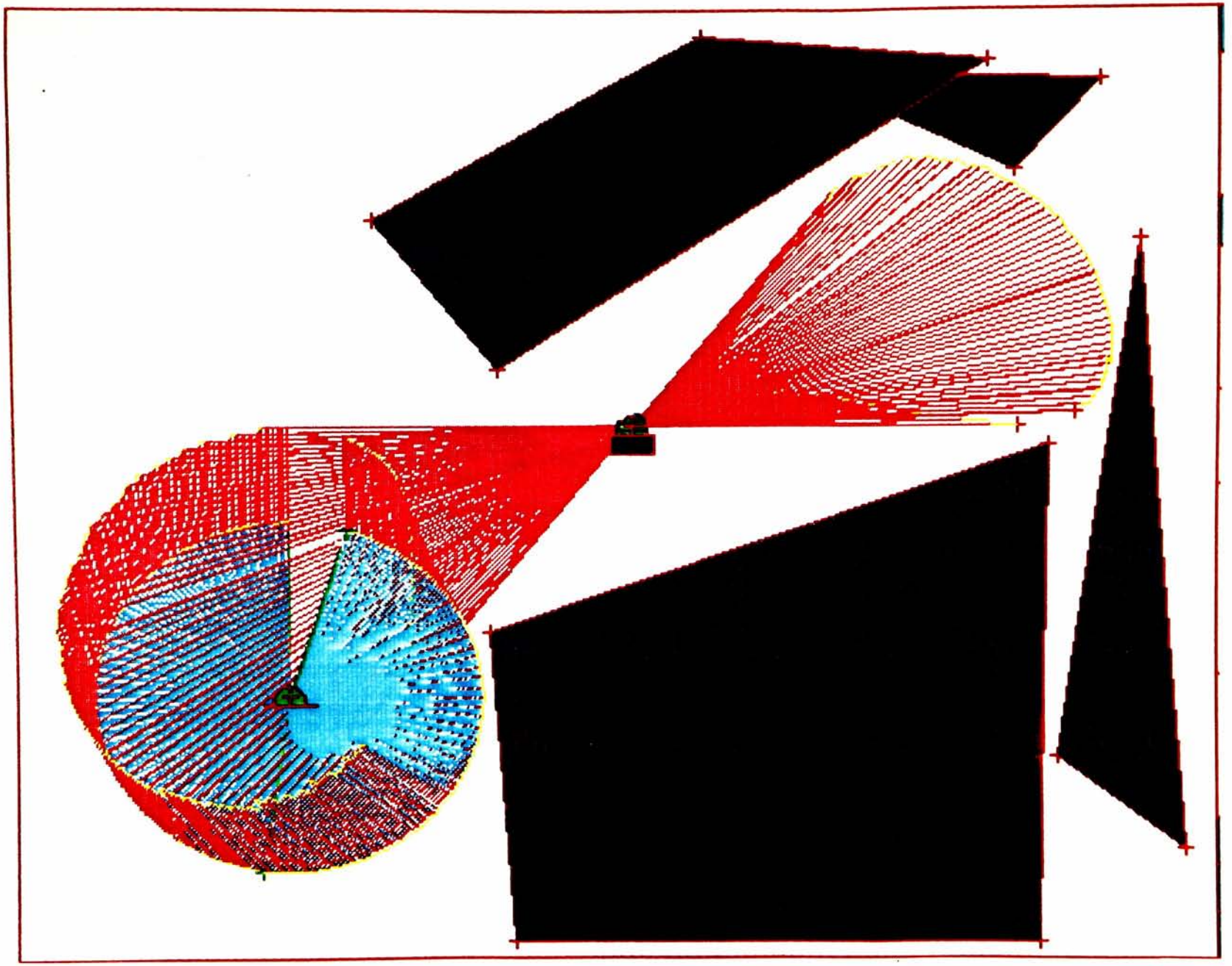


Figure 3-1 Rotating Driver with Rotating Guide, in Motion, 1 degree rotation increment, Trace "on".

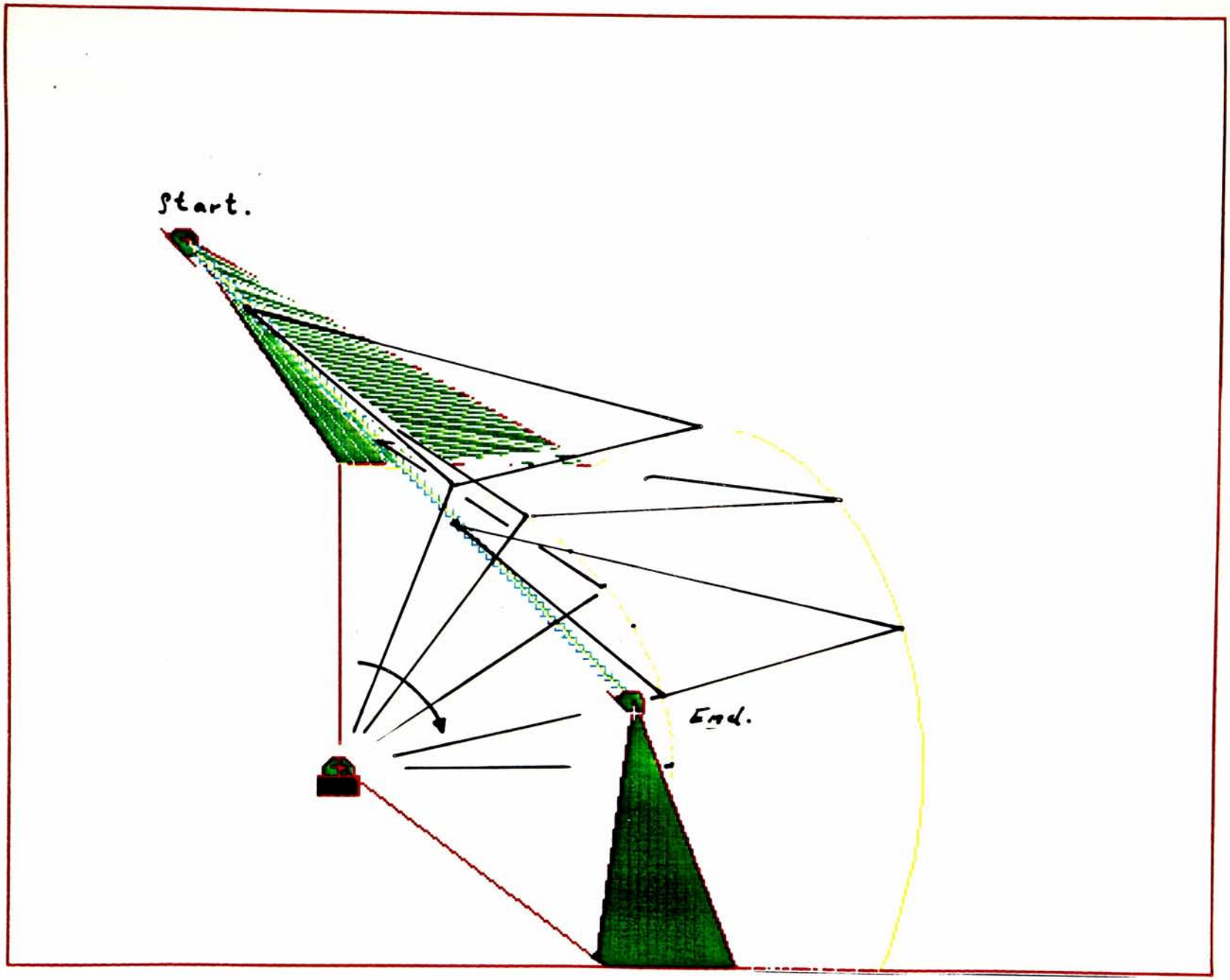


Figure 3-2 Translating Driver with Dyad-2, in Motion, Trace "on".

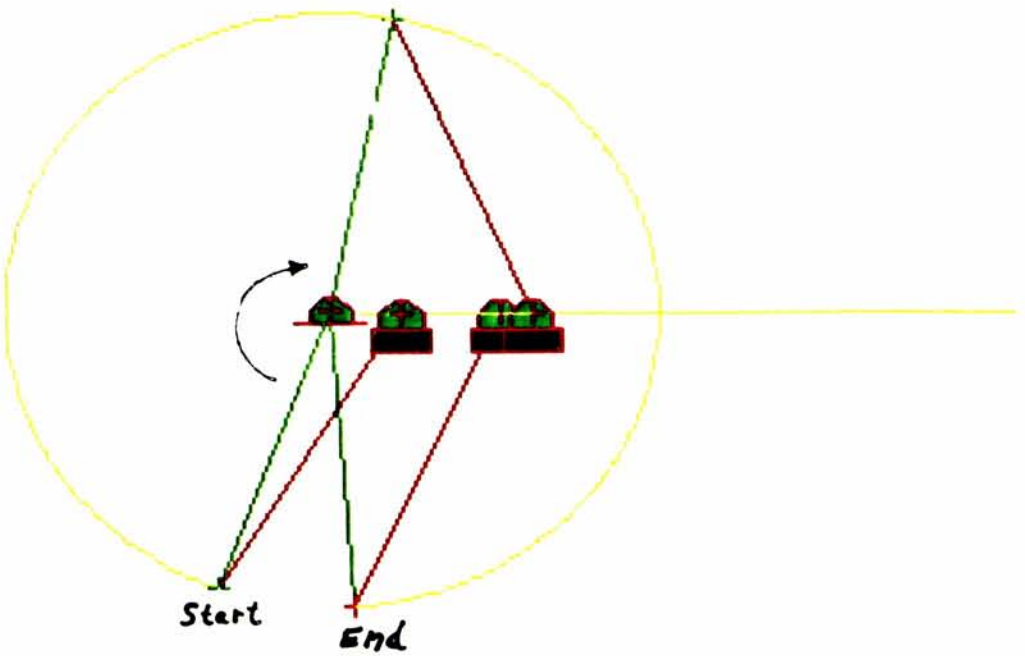


Figure 3-3 Rotating Driver with Slider (Crank-Slider), in Motion, Trace "on", Only Tracer Path Displayed.

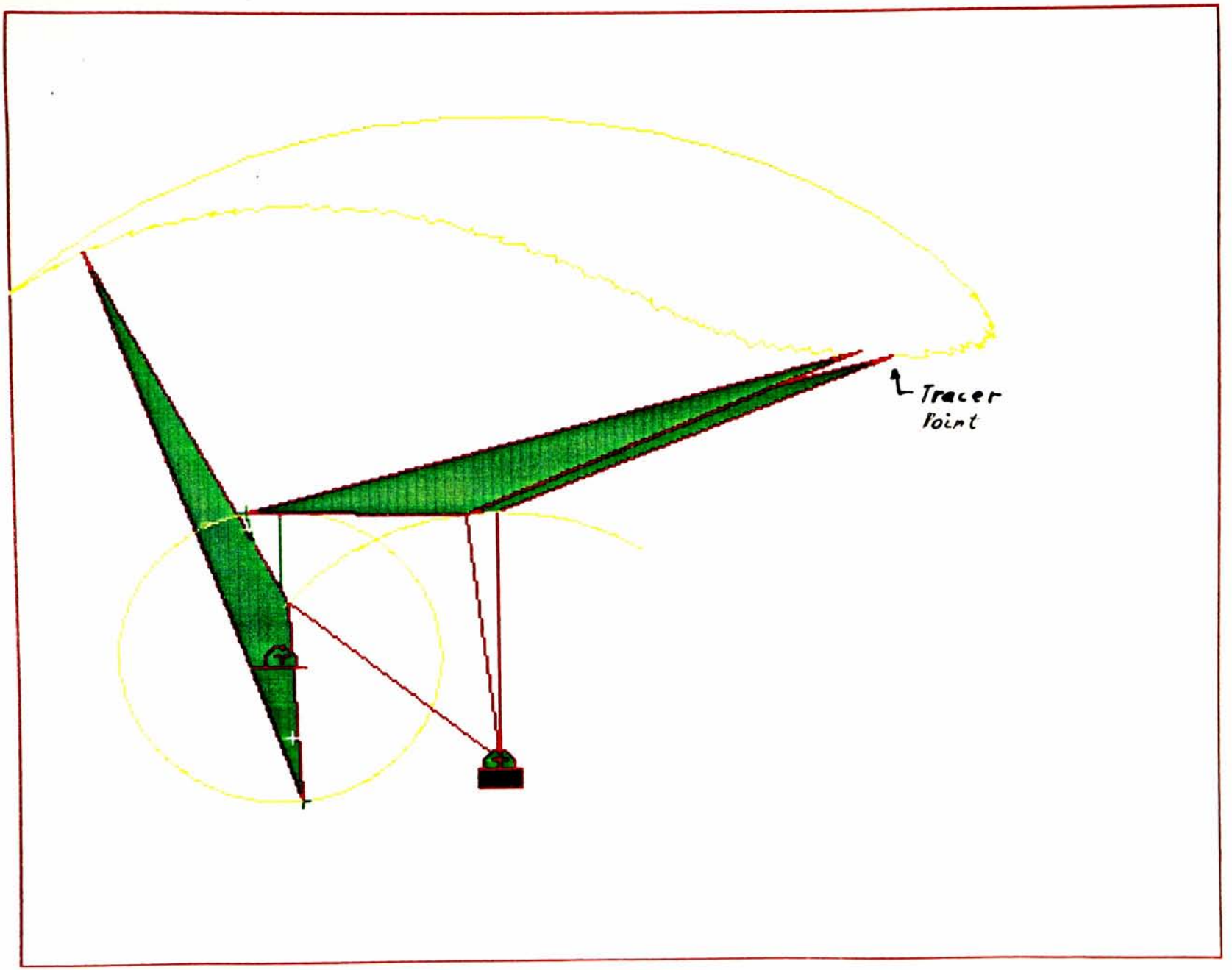


Figure 3-4 Rotating Driver with Dyad-2, in Motion, Trace "on", Only Tracer Path Displayed.

point is clearly obvious. In Figure 3-4 the "jagged" appearance of the trace is apparently due to the resolution of the computation (1 degree) and the resolution of the CRT.

Another example of the above mentioned animation display technique, in which the actual behavior of the linkage is not lost, is shown in Figure 3-5. In this case, not only is the path of the tracer point obvious, but also the motion of the rotating guide causing that path is clear. Figure 3-6 has three components, but it is similar to the case above (Figure 3-5) in that motion is obvious from the traces and rendering of the components at their various positions. It seems that there are cases where the user might want the capability of having the motion displayed either way. This capability is recommended for future upgrades to this system.

In certain linkages, particularly those including dyads, the mode of assembly (i.e. the location of one leg relative to another) of the dyad can reverse during motion. An example is offered in Figure 3-7. In this figure, a rotating driver is connected to a simple dyad, and the linkage is originally at position 1. In this original configuration, the dyad looks like a backward "L." Motion begins, and the driver moves clockwise. The dyad follows, and reaches a maximum rightmost position at point 2. As the driver rotates further around, the dyad straightens out until finally, at point 3, the dyad reaches its binding position, and the driver can proceed no further.

As currently designed, LKSP stops motion at this point, and displays "Linkage Binding." Motion of the driver requested in the reverse direction is honored, but the dyad would retrace the same path as described above in reverse order. It is sometimes (but not always) desirable to reverse the mode of assembly when the dyad reaches point 3 (the so-called toggle point) in order to conveniently view the remainder of the motion cycle. Figure 3-7 was created from an earlier version of LKSP in which the mode of assembly for dyads was reversed whenever the dyad reached a binding position. At point 4, the mode of assembly reverses, and, as the driver returns to its original position and beyond, the dyad passes through points 5 and 6. At point 7, the dyad binds again.

An example of a case in which reversal of the mode of assembly would be desirable is shown in Figure 3-8. This figure illustrates a rotating driver connected to a dyad-2 with an external path tracer point located at an extreme distance from the linkage. As the driver rotates clockwise from position 1 to position 2, the dyad-2 follows, and the tracer point exhibits a potentially interesting path. However, visualization of the rest of the path requires reversal of the mode of assembly. This suggests that it

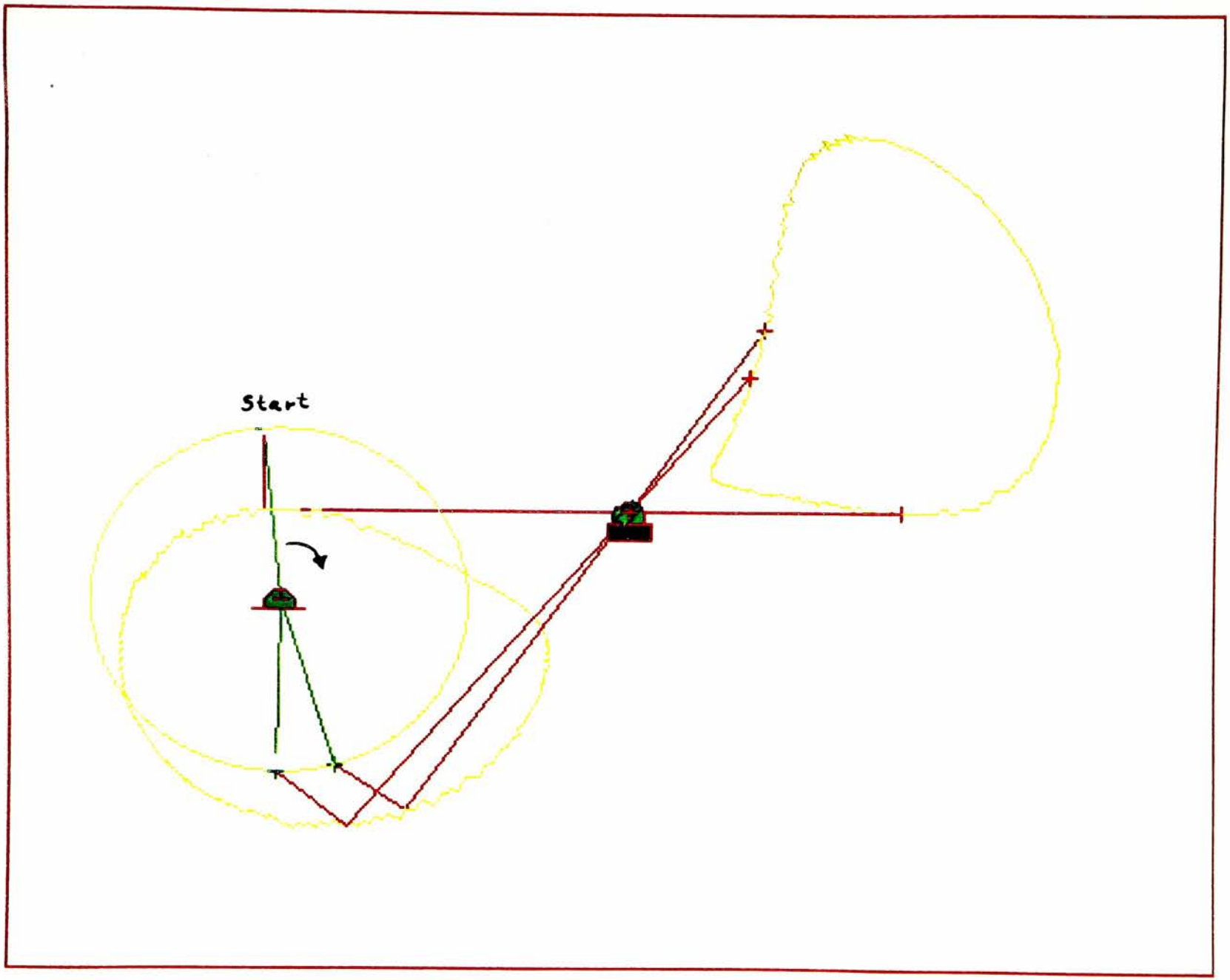


Figure 3-5 Rotating Driver with Rotating Guide, in Motion, Trace "on", Only Tracer Path Displayed.

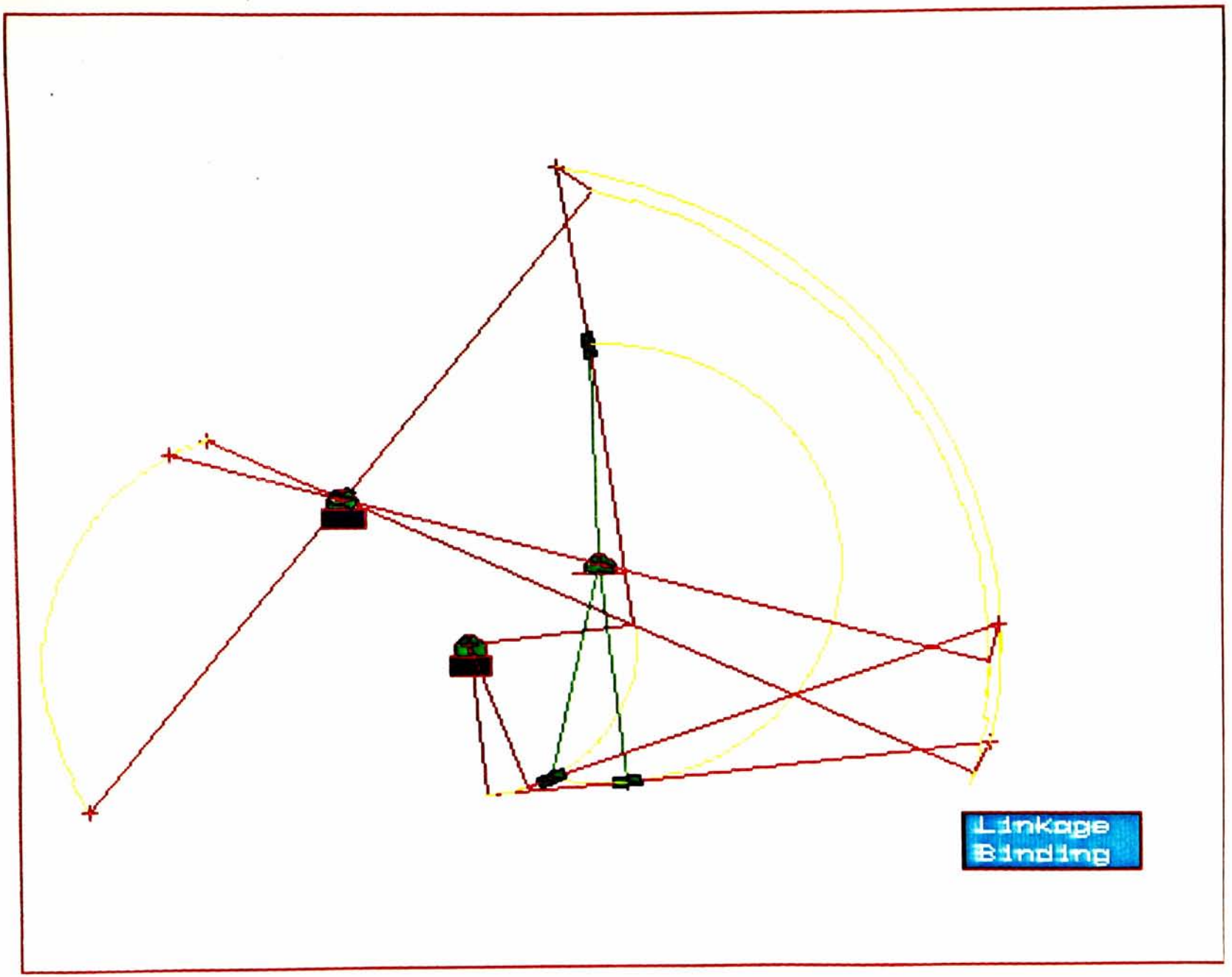


Figure 3-6 Rotating Driver with Oscillating Slider and Rotating Guide, in Motion, Trace "on", Only Tracer Path Displayed. 3-10

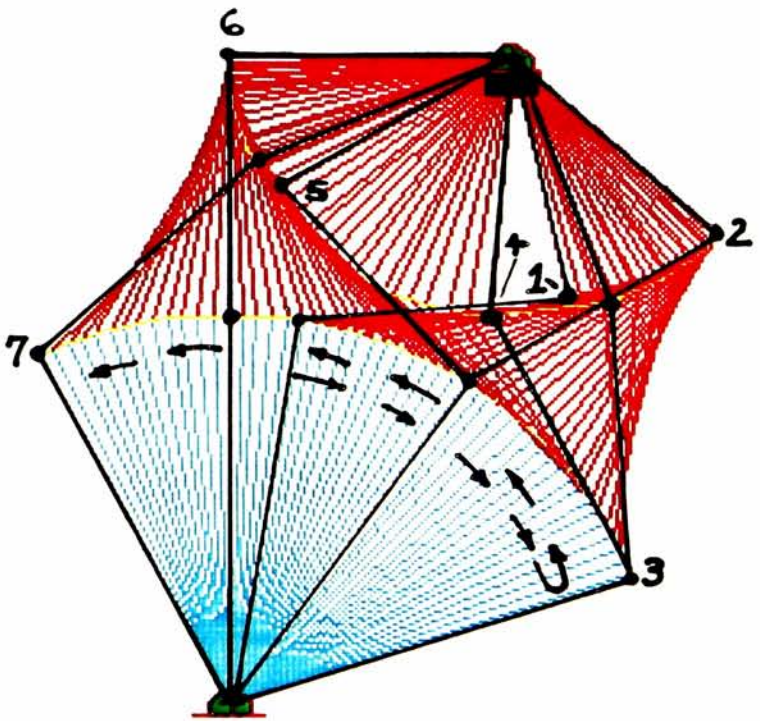


Figure 3-7 Rotating Driver with Dyad-1, in Motion, Trace "on", Mode of Assembly Reverses.

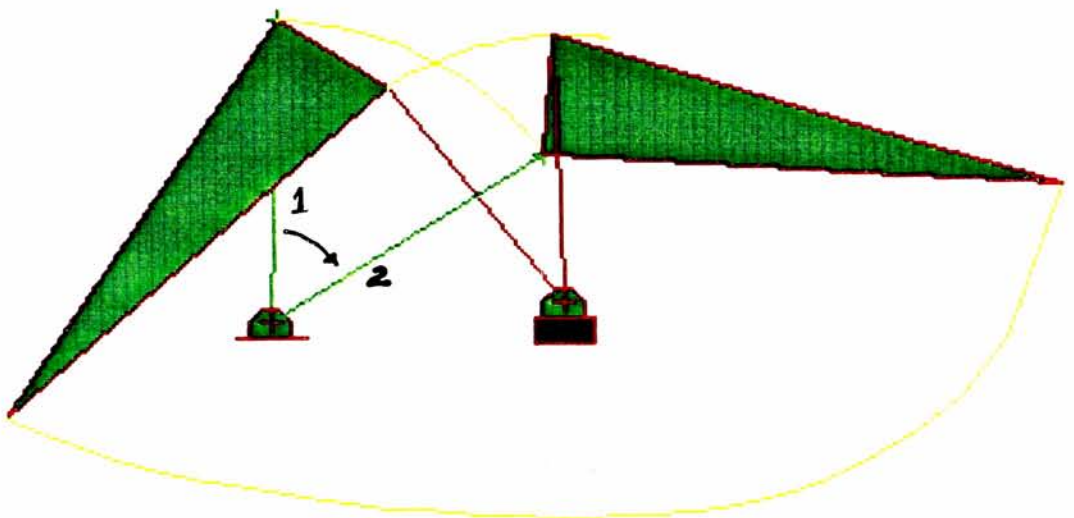


Figure 3-8 Rotating Driver with Dyad-2, in Motion, Trace "on", Only Tracer Path Displayed, Linkage Binding.

is probably desirable, for future upgrades of LKSP, to include a user controlled capability to reverse the mode of assembly of dyads at binding.

It is well to note that this minimum system has received some measure of success from the standpoint of user/computer interface and desirable features. Any future upgrades including extension of the currently available features would require multiple menus and a system to keep track of command phases.

As a final point, it was generally agreed that the limit of animation comprehensibility was reached at about five linkage components. It is believed that the limiting factor is the screen size of the CRT. More components can be handled if they are small, but the screen resolution then becomes a limiting factor.

3.3 Operating Efficiency

System response could not be controlled, but most runs were made during low usage hours (after 6:00 PM, or on weekends) so system response was always excellent. In all cases, the animation speed was adequate. The more components in the linkage, the slower the animation speed. On a busy timesharing system, the animation can be expected to slow considerably. For implementation of LKSP on a slower personal computer, response could be dealt with via deviation from ACM core standard graphics with more device dependent algorithms.

3.4 The Hardware Used

Inspection of the color plots in this report makes it apparent that the Tektronix 4109 display is somewhat jagged. Normally, this is not a serious problem, but it can be bothersome at times. Use of anti-aliasing would obviously help, but would also slow down the display due to the added complexity of the scan conversion algorithm. A display with higher resolution (1040 by 1280 is readily available today) would greatly improve the display and the level of complexity that could be handled by the user.

LKSP was run on the Tektronix terminal, connected to the VAX via RS-232 on a dedicated line at 9600 baud. This transmission speed is considered a minimum for graphics applications. Some runs were made on a broadband network at 19200 baud with no discernible difference in the appearance of the animation. The broadband runs were made during the day when the VAX system was probably busier than the evening and weekends when the runs on the dedicated line were made. Also, the broadband

network requires more overhead than the RS-232 line.

The Tektronix 4695 ink-jet plotter, which was used to create the original color graphics for the figures in this report, produces a close facsimile of the screen graphics, but the colors are not exactly the same as screen colors. In addition, other variations from the original colors occur in the duplication of the figures. Finally, colors were discernibly different in appearance on the various Tektronix terminals used.

The VMS operating system and VAX Pascal were very easy to use, and the documentation was very good. Also, the VMS Pascal software development environment was found to be user friendly.

3.5 Suggestions for Future Extensions

A number of features have already been mentioned in previous sections, namely,

1. User selects method of tracing motion on screen.
2. User selects motion step sizes (or defaults).
3. User has ability to shift mode of assembly.

In addition, the capability for a more exact input technique (e.g. keyboard input of coordinates or some combination of coordinates, lengths, angles, etc.) would be desirable for a future system. Also, the ability to save and restart a linkage (originally proposed for this system as a desirable future upgrade) was clearly stated to be a useful feature for future versions of LKSP.

Other features which were determined to be potentially useful were the ability of the user to select alternate color schemes for hard copies or screen display and the incorporation of a component modification feature. This concludes discussion of the evaluation of LKSP, conclusions and recommendations.

4. Bibliography.

4. Bibliography.

1. Reuleaux, Kinematics of Machinery, a translation of Reuleaux, F., Theoreitische Kinematik, Friedrich Vieweg und Sohn, Brunswick, Germany, 1875, by A. B. W. Kennedy, Macmillan, London, 1876, reprinted by Dover, New York, 1963.

2. Kinematics and Mechanisms Design, Chung-Ha Suh and Charles W. Radcliffe, 1978, John Wiley and Sons.

3. Linkage Design Monographs, Part of Final Report on NSF grant GK-36624, A. H. Soni, Editor, 1974, Pub. by Oklahoma State University.

4. "Mechanism Design Using Animation," H. E. Ladd, Transactions of the ASME, Vol. 98, Ser B, No. 4, November 1976.

5. Private Communication, Mr. R. Curtin, Library Director. Engineering Division Library, Eastman Kodak Company, Kodak Park Division, Rochester, New York.

6. Introduction to Robotics, Arthur J. Critchlow, 1985, MacMillan.

7. Analysis of Mechanisms and Robot Manipulators, Joseph Duffy, 1980, John Wiley and Sons.

8. "Kinematic Equations for Simple Manipulators," R. P. Paul, B. Shimano and G. E. Mayer, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No. 6, June 1981.

9. "Anatomy of Industrial Robots and Their Control," J. Y. S. Luh, IEEE Transactions on Automatic Controls, Vol. 28, February 1983.

10. "Robot Arm Kinematics, Dynamics and Control," C. S. G. Lee, Computer, IEEE, Vol. 15, No. 12, Dec. 1982.

11. "Robot Kinematics Using A Microcomputer," H. W. Townes, D. O. Blackketter and J. W. Lowell, Proceedings of the Society for Computer Simulation (SCS), Conference on Modeling and Simulation with Microcomputers, January 1982.

12. "A System Approach to Dynamic Simulation of Robotic Manipulators," Y. Stepanenko and T. S. Sankar, Computers in Mechanical Engineering, CIME, Vol. 3, No. 6, May 1985.

13. Kinematics and Linkage Design, A. S. Hall, Jr., 1961, Prentice-Hall.

14. Applied Linkage Synthesis, D. C. Tao, 1964, Addison-Wesley.

15. "An Iterative Method for the Displacement of Spatial Mechanisms," John Uicker, Jr., J. Denavit and R. S. Hartenberg, Journal of Applied Mechanics, Transactions of the ASME, Paper No. 63-WA-45, March 1963.

16. "Velocity, Acceleration and Static-Force Analysis of Spatial Linkages," J. Denavit, R. S. Hartenberg, R. Razi and J. J. Uicker, Jr. ASME Paper No. 65-APMW-13, published by ASME, 1965.

17. "Dynamic Force Analysis of Spatial Linkages," J. J. Uicker, Jr. ASME Paper No. 66-Mech-1, published by ASME, 1966.

18. "Dynamic Behavior of Spatial Linkages," Part 1 Exact Equations of Motion, Part 2 Small Oscillations About Equilibrium, J. J. Uicker, Jr. ASME Paper No. 68-Mech-5, published by ASME, 1968.

19. Matrix Methods in the Design and Analysis of Mechanisms, John J. Uicker, Jr., 1970, unpublished set of notes for course taught by the author at the University of Wisconsin.

20. "KIDYAN, Computer-Aided Kinematic and Dynamic Analysis of Planer Mechanisms," V. Brat and P. Lederer, Mechanism and Machine Theory, Vol. 8, No. 4, 1973.

21. "Self Generating Analysis Algorithm for Computer Based Design of Mechanisms," W. S. Reed and R. E. Garrett, ASME Paper No. 74-DET-65, published by ASME, 1974.

22. "Computer Aided Analysis of Kinematic Mechanisms," I. H. Gould and P. C. Ingham, Proceedings of the International Conference on Interactive Techniques in Computer Aided Design, Bologna, Italy, published by IEEE Computer Society, 1978.

23. "Contribution of the Forming of Computer methods for Automatic Modeling of Spatial Mechanisms," Parts 1 and 2, M. Vukobratovic and V. Putkonjak, Mechanism and Machine Theory, Vol. 14, No. 3, 1979.

24. "Derivation of General Kinematic Equations of Spatial Constrained Mechanical Systems with the Aid of a Computer," V. Brat, V. Stejskal and F. Opicka, Mechanism and Machine Theory, Vol. 14, No. 5, 1979.

25. "On the Dynamic Simulation of Large Nonlinear Mechanical Systems: Part 1 An Overview of the Simulation Technique. Substructuring and Frequency Domain Considerations," R. J. Cipra and J. J. Uicker, Jr. ASME Paper No. 80-DET-66, published by ASME, 1980.

26. "On the Dynamic Simulation of Large Nonlinear Mechanical Systems: Part 2 The Time Integration Technique and Time Response Loop," R. J. Cipra and J. J. Uicker, Jr. ASME Paper No. 80-DET-67, published by ASME, 1980.

27. "Computer Method for Kinematic Analysis of Lower Pair Mechanisms 2.," J. G. de Jalon, M. A. Serna, R. Aviles. Mechanism and Machine Theory, Vol. 16, No. 5, 1981.

28. "Computer Method for Kinematic Analysis of Lower Pair Mechanisms 1.," J. G. de Jalon, M. A. Serna, R. Aviles. Mechanism and Machine Theory, Vol. 16, No. 5, 1981.

29. "Mathematical Models for Analysis and Synthesis of Spatial Mechanisms," I, R. I. Alizade, J. Duffy and E. T. Hajiyev, Mechanism and Machine Theory, Vol. 18, No. 5, 1983.

30. "Vector Network Models for Kinematics: The 4-Bar Mechanism," K. Singhal, H. K. Kesavan and Z. I. Ahmad, Mechanism and Machine Theory, Vol. 18, No. 5, 1983.

31. "The Development of a Suite of Programs for the Analysis of Mechanisms," A. J. Medland, Digital Systems in Industrial Automation, Vol. 2, No. 1, 1983.

32. "Modeling of Dynamic Mechanical Systems," Milton A. Chace, paper contributed to the CAD/CAM Robotics and Automation Institute and International Conference, February 1985, Tucson, Arizona.

33. "IMP (Integrated Mechanisms Program), A Computer-Aided Design Analysis System for Mechanisms and Linkage," P. N. Sheth and J. J. Uicker, Jr. ASME Paper No. 71-Vibr-80, published by ASME, 1971.

34. "A Generalized Symbolic Notation for Mechanisms," P. N. Sheth and J. J. Uicker, Jr. ASME Paper No. 70-Mech-19, published by ASME, 1970.

35. "A method for the Identification and Recognition of Equivalence of Kinematic Chains," J. J. Uicker, Jr., Mechanism and Machine Theory, Vol. 10, 1975.

36. "Computer-Aided Design of precision Control Linkages in the Classroom," K. W. Chase, ASME Paper No. 80-A/DSC-39, published by ASME, 1980.

37. "Kinematic and Dynamic Analysis of Mechanisms by Programmable Calculator," J. M. Prentis, International Journal of Mechanical Engineering Education, Vol. 9, No. 1, 1981.

38. "Survey of Computer Use in Mechanism Analysis and Synthesis, Proceedings of the ASME Design Engineering Technical Conference," published by ASME, 1974.

39. "Mechanism Design by Computer," Roger E. Kaufman, Machine Design, Vol. 50, No. 24, 1978.

40. "KINSYN Phase II: A Human-Engineered Computer System for Kinematic Design and A New Least Squares Synthesis Operator," R. E. Kaufman, Mechanism and Machine Theory. Vol. 8, No. 4, 1973.

41. "KINSYN III: A New Human-Engineered System for Interactive Computer Aided Design of Planer Linkages," A. J. Rubel and R. E. Kaufman, Journal of Engineering for Industry, Transactions of the ASME, Vol. 99, Ser B, No. 2, May 1977.

42. "LINCAGES, Linkage Interactive Computer Analysis and Graphically Enhanced Synthesis Package," A. G. Erdman and J. E. Gustafson, ASME Paper No. 77-DET-5, published by ASME, 1977.

43. "Computer-Aided Linkage Design Using the LINCAGES Package." A. G. Erdman and D. R. Riley, ASME Paper No. 81-DET 121, published by ASME, 1981.

44. "Analyzing Mechanisms with the IBM PC," Clark R. Baker, Computers in Mechanical Engineering, CIME, Vol. 2, No. 1, July 1983.

45. "Simplified Data Structure for Analysing Mechanisms Using Character Handling Techniques," M. R. Smith and Z. Ye, Computer Aided Design, Vol. 16, No. 4, July 1985.

46. "New Directions for Mechanism Kinematics and Dynamics." Arthur G. Erdman and Donald F. Riley, Computers in Mechanical Engineering, CIME, Vol. 3 No. 6, May 1985.

47. DADS Users Manual, Rev. 3.0, Preliminary, Computer Aided Design Software, Inc. (CADSi), 1985, CADSi.

48. "The Dynamic Duo: Dram and Adams", Gary Dawson, Computers in Mechanical Engineering, CIME, Vol. 3 No. 5, March 1985.

49. "Computer-Aided Design of Mechanisms: 1984 and Beyond," A. G. Erdman, Mechanism and Machine Theory, Vol. 20, No. 4, 1985.

50. Mechanisms for Engineering Design, Stanley B. Tuttle, 1967, John Wiley and Sons.

51. "Algorithm for Automatic Sketching of Planer Kinematic Chains," D. G. Olson, T. R. Thompson, D. R. Riley and A. G. Erdman, Journal of Mechanisms, Transmissons and Automation in Design, Vol. 107, No. 1, 1985.

52. "Integrated CAD of Mechanisms," W. A. Mittelstadt, D. R. Riley, and A. G. Erdman, Mechanism and Machine Theory, Vol. 20, No. 4, 1985.

53. "Status Report of the Graphic Standards Planning Committee of ACM," Edited by R. Heilman and B Herzog, Computer Graphics, a Quarterly Report of SIGGRAPH-ACM (Special Interest Group, Graphics, the Association for Computing Machinery), published by ACM, Vol. 13, No. 4, 1979.

54. Fundamentals of Interactive Computer Graphics, J. D. Foley and A. Van Dam, 1982, Addison-Wesley.

55. Computer Graphics, A Programming Approach, Steven Harrington, 1983, McGraw-Hill.

56. VAX/VMS, version 4.0, documentation, Volume 1., General Information, September, 1984, Digital Equipment Corporation, Maynard, Mass.

57. TEK, Programmers Reference Manual, 4107/4109 Computer Display Terminals, September, 1983, Tektronix, Inc., Beaverton, Oregon.

58. TEK, Service Manual, 4109/CX Computer Display Terminal, October, 1985, Tektronix, Inc., Beaverton, Oregon.

59. TEK, Operators Manual, 4107/4109 Computer Display Terminals, 1983, Tektronix, Inc., Beaverton, Oregon.

60. TEK, Reference Guide, 4107/4109 Computer Display Terminals, 1983, Tektronix, Inc., Beaverton, Oregon.

61. Pascal User Manual and Report, Kathleen Jensen and Niklaus Wirth, second edition, 1978, Springer-Verlag.

62. Programming in VAX Pascal, March, 1985, Digital Equipment Corporation, Maynard, Mass.

63. VAX Pascal, User's Guide, 1985, Digital Equipment Corporation, Maynard, Mass.

Appendix I Glossary

Glossary.

dynamic load	A force produced by or from motion, (e.g. inertia of a disk head on an arm).
four-bar linkage	Looped linkage in which there are three connected links. Two end links are grounded and the ground forms the "fourth" bar or link (Figure 1 1).
function generation linkage	A linkage designed primarily to precisely guide the movement of a rigid body through a path defined by some functional relation, (e.g. throttle linkage).
guidance linkage	A linkage designed primarily to guide the movement of a rigid body (e.g. a bucket loader).
impact load	A very sudden dynamic load produced by the meeting of bodies in motion, (e.g. a punch press).
kinematics	From the Greek kinein, to move, i.e. related to linkage motion.
kinematic analysis	Analysis aimed at an understanding of linkage motion.
kinetics	From the Greek kinetikos, produced by movement, (i.e. related to forces producing or produced by motion).
kinetic analysis	Analysis of forces producing, or produced by motion, e.g. inertial forces.
linkage	A system of "links" or bodies connected together in which the motion of one body (or more than one) influences the motion of the others in some desired fashion, (e.g. Figure 1 1).

looped linkage	A linkage in which the connected bodies or links form a loop (Figure 1-1).
mechanism	"A combination of rigid or resistant bodies so formed and connected that they move upon each other with definite relative motion", Reuleaux (1).
machine	A combination of one or more mechanisms, typically configured to work together to perform some overall function(s).
nonlooped linkage	A linkage in which the connected bodies or links does not form a loop (Figure 1 1).
path generation linkage	A linkage designed primarily to precisely guide the movement of a rigid body through a predetermined path (e.g. a sewing machine linkage to drive the needle).
planar linkage	A linkage in which all motion is in one plane or in parallel planes (2-D).
slider-crank	Looped linkage in which there are two connected links. One rotating driver moves a connecting link which slides on ground (Figure 1 1).
spatial linkage	A linkage in which motion in three spatial directions can occur (3-D).
static load	A force produced by or from very gradual motion, (e.g. a scissors jack). or a non-moving load such as weight.
synthesis	The reverse of analysis wherein a design objective is specified and the geometry of the linkage is calculated. The linkage is "synthesized" through a set of rules to meet the specification within some predetermined limit. Normally, a specific type of linkage (e.g. a 4-bar) is considered to limit the search. There are normally two synthesis steps, "type synthesis" wherein the category of the proposed linkage is determioned, and "dimensional synthesis" in which actual dimensions are specified.

Appendix II Listing, Main Program.

```
{                        Mike St. Jacques          }
{                File Name, lkspm.pas              }

[ ENVIRONMENT( 'ENVGR.ENV' ) ] Program LKSPM( input, output );


%INCLUDE 'LKTY.pas/NOLIST'       {  LKTY.pas = const, type, var declarations }

%INCLUDE 'GREXT.pas/NOLIST'      {  GREXT.pas = external declarations for    }
                                 {  graphics procedures   }

%INCLUDE 'LKICEXT.pas/NOLIST'    {  LKICEXT.pas = external declarations for  }
                                 {  graphics icons procedures   }

%INCLUDE 'LKPREPEXT.pas/NOLIST' {  LKPREPEXT.pas = external declarations for }
                                 {  prep procedures   }

%INCLUDE 'LKDO1EXT.pas/NOLIST'   {  LKDO1EXT.pas = external declarations for }
                                 {  DO procedures   }

%INCLUDE 'LKDO2EXT.pas/NOLIST'   {  LKDO2EXT.pas = external declarations for }
                                 {  more DO procedures   }

%INCLUDE 'LKDO3EXT.pas/NOLIST'   {  LKDO3EXT.pas = external declarations for }
                                 {  more DO procedures   }

%INCLUDE 'LKDO4EXT.pas/NOLIST'   {  LKDO4EXT.pas = external declarations for }
                                 {  more DO procedures   }

%INCLUDE 'LKTRIGEXT.pas/NOLIST' {  LKTRIGEXT.pas = external declarations }
                                { for Trig procedures   }
```

```
begin  { LKSPmain }                              { main operating program   }

        SetupTek;
        Prep;                                    { handles all prelim    }
        GetParameters;

        repeat
                GetMenuItem( pickedMenuItem );

                if ( pickedMenuItem = errorpick ) then doerrorpick

                else if ( pickedMenuItem = MGridf ) then dogridf

                else if ( pickedMenuItem = MGridc ) then dogridc

                else if (pickedMenuItem = MRotatingDriver) then doRotatingDriver

                else if ( pickedMenuItem = MTransDriver) then doTransDriver

                else if ( pickedMenuItem = MReplaceDriver ) then doReplaceDriver

                else if ( pickedMenuItem = MDyad1 ) then doDyad1

                else if ( pickedMenuItem = MDyad2 ) then doDyad2

                else if ( pickedMenuItem = MOscSlider ) then doOscSlider

                else if ( pickedMenuItem = MRotatingGuide ) then doRotatingGuide

                else if ( pickedMenuItem = MSlider) then doSlider

                else if ( pickedMenuItem = MIpanel3 ) then doIpanel3

                else if ( pickedMenuItem = MIpanel4 ) then doIpanel4

                else if ( pickedMenuItem = MDelete ) then doDelete

                else if ( pickedMenuItem = MPickt ) then doPickt

                else if ( pickedMenuItem = Mmove ) then doMove

                else if ( pickedMenuItem = MCleanup ) then doCleanup

        until( pickedMenuItem = MEndSession );

        DisplayFinishUp;

end.
```

Appendix III Listing, Prep Module.

```
        {       Mike St. Jacques        }
        { file name = LKPREP.pas        }


[ INHERIT( 'ENVGR.ENV' ) ] module lkprep( input, output );

{ file ENVGR.ENV is the environment file created by LKSPM.pas }
{ ie. module lkprep shares the environment, types, consts etc   }



[ GLOBAL ] procedure Prep;

begin { Prep }

        InitializeVariables;

        makeInitGraphics;
        DisplayInitGraphics;

        SetWindow( LKWindowxlo, LKWindowxhi, LKWindowylo, LKWindowyhi );
        Viewport( 0.0, 1.0, 0.0, 1.0 );

        makeFinishUp;
        makeBorder1( red );
        makeBorder2( red );
        makeGridc( Lgrey );
        makeGridf( Lgrey );
        makeGin1Enable( 5.0, 5.0 );
        makeGin2Enable( on, on, 5.0, 5.0 );
        makeGin3Enable( on, on, 5.0, 5.0 );
        makeGin4Enable( on, on, 5.0, 5.0 );
        makeGin5Enable( on, on, 5.0, 5.0 );
        makeGreeting( 2.0, 9.0 );
        makeLBmessage;
        maketrashLBmessage;
        makeClearScreen;
        makeMenu( red, Lgrey, blue );
        makeAccMenu( red, BLgrey, blue );

        DisplayBorder1;
        DisplayBorder2;
        DisplayGreeting;
        DisplayMenu;

end;  { Prep }
```

Appendix IV

Device Driver and Core Graphics Procedures.

```
        {       Mike St. Jacques                                }
        { file name = grtek2.pas, device driver                 }
        { for Tektronix 4109 & ACM core graphics procedures }

[ INHERIT( 'ENVGR.ENV' ) ] module grtek2( input, output );

{ file ENVGR.ENV is the environment file created by LKSP.pas }
{ ie. module grtek2 shares the environment, types, consts etc}

CreateSeg( usegno :integer );
CloseSeg;
SetVis( usegno : integer; v : vistype );
RenameSeg( oldname, newname :integer );
DeleteSeg( usegno : integer );

update;

SetGinDispStartPt( x, y : real );
GinEnable( noGinpoints: noGinpointstype );
getGinxy( var xout, yout: real );
GinDisable;
SetGinInking( inkingmode: switchtype);
SetGinRubberbanding( rubingmode: switchtype);
SelectPointMarkerType( Markerno : Markernotype );
SelectPanelFillColor( c: colortype );
SelectPanelFillPattern( FillPatternno : FillPatternnotype );
BeginPanelBoundary( x, y : real );
EndPanel;

Initmatrix( var matrixin: cmatrixtype );
TransCTMAbs2( tx, ty : real );
TransCTMRel2( tx, ty : real );
ScaleCTMAbs2( sx, sy : real );
ScaleCTMRel2( sx, sy : real );
RotateCTMAbs2( angle : real );
RotateCTMRel2( angle : real );
SetCTMAbs2( matrixin : cmatrixtype );
SetCTMRel2( matrixin : cmatrixtype );
RotateAboutPt( x, y, angle : real );

SetWindow( xmin, xmax, ymin, ymax : real );
Viewport( xmin, xmax, ymin, ymax : real );

SetLinestyle( s : lstyletype );
SetColor( c : colortype );
SetTextColor( c : colortype );

LineAbs2( x, y : real );
LineRel2( dx, dy : real );
MoveAbs2( x, y : real );
MoveRel2( dx, dy : real );
PointAbs2( x, y : real );
PointRel2( dx, dy : real );
mText( m : messagetype );

ClearScreen;
Initialize;
Terminate;
```